FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Estimating Wind Turbine Generator Failures Using Machine Learning

João Miguel Guimarães Fidalgo Roque

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof.Luis Guimarães

July 23, 2017

Resumo

Esta tese resulta da implementação de um algoritmo de machine learning, com o objectivo de prever falhas em geradores de turbinas éolicas. Esta tese usa dados referentes ao ambiente da turbina, como velocidade média do vento, produção anual, tipo de terreno, modelo da turbina, etc, em conjugação com um historial de falhas conhecidas, de modo a prever a quantidade de falhas para o próximo ano, por parque e por tecnologia. A rede neuronal implementada foi uma rede numa arquitectura feed-forward não linear. Este projecto segue o processo, desde a transformação de dados não tratados até à implementação do algoritmo.

Abstract

This thesis is an implementation of a machine learning algorithm, in order to predict failures in wind turbine generators. The thesis uses data relating to wind turbine failures and their context, average wind speed, annual production, type of terrain, wind turbine generator model, etc, in order to predict the amount of failures in the next year, per park and per technology. A neural network in a feed-forward non linear architecture was used. This project follows the transformation and implementation of the network from raw data, to the training parameters and implementation of the network.

Acknowledgments

I would like to thank my family, my friends and my teachers.

João Roque

"To read without reflecting is like eating without digesting."

Edmund Burke

Contents

List of Figures

List of Tables

Abreviaturas e Símbolos

WTG	Wind turbine generator
MSE	Mean squared error
RMSE	Root mean squared error
SVM	Support vector machines
MLP	Multilayer perceptron

Chapter 1

Introduction

In this chapter, the context Section 1.1, motivation, Section 1.2 and structure of the dissertation, Section 1.3, are explained.

1.1 Context

This thesis exists in the context of the last semester of the course, Master in Electrical and Computers Engineering in the faculty of engineering of the university of Porto.

Nowadays, one of the key trends of industry is the shift towards the notions of interconnectivity, of industry 4.0, of internet of things. This paradigm allows for a continuous feed of real data coming from machines and equipments, thus creating a new challenge: what to do with that data. This new paradigm is allowing companies to take a new perspective in their day to day challenges. Among these challenges one that is ever present in all aspects of industry is the notion of reliability and of fallibility, which is elemental when maintaining a healthy, efficient and sustainable production. Knowing when something will fail, learning what is most likely asset to fail is critical in increasing efficiency.

The answer to this topics is introduced in this thesis in the form of a practical problem. Doing maintenance to wind turbines, specially in high sea or in a harsh environment is very costly. It involves the deployment of a large number of resources. So predicting when it will fail is a very important information when planning and optimizing maintenance.

Renewable energy is a big mega trend, and if one pictures a leading company with tens of Gigawatts of installed production, with international implementation, operation and maintenance (O&M) costs are a growing concern.

WTG components (gearbox, generator, blades, etc) have a 20 year designed life cycle and an important part of wind farms are approaching that age. Thus maintenance planning has a growing impact in O&M costs.

An unplanned maintenance has several costs, some obvious as the loss of power production, but others less obvious, as the materials required for maintenance and repairs are not always available, and are highly specialized. Thus an unplanned maintenance forces a chain of costs. Therefore, the ability to predict the need for maintenance would be relevant in decreasing costs. During the years of production, there was a gathering of data regarding the functioning of the WTG. So, there seems to be room for a starting to implement a system of preventive maintenance, where one can predict when the turbine is likely to fail and schedule the maintenance with more accuracy and a optimum allocation of resources.

1.2 Motivation

Optimization is a key source of advancement in our society. Machine learning algorithms, as this dissertation shows, are not just a theoretical consideration, but can create a real impact and change in an organization. They are very important as ways to triage information and to harness useful data and as a way to draw informed decisions and planning. The evolution of computational power, turns this once theoretical prepositions as vital ways to deal with data.

In a very competitive environment, such as the energy production sector, the gain in efficiency is an industry focal point, and seems to have a great space for innovation. Another important point is the relation between decreasing O&M costs and increasing the potential of the technology. As any business if the operational costs are diminished, that grows space for implementing more wind turbines and increasing a renewable energy efficiency.

1.3 Objectives

The aim of this dissertation is to apply a machine learning solution to predict wind turbine generator failures. This work comes as a response to a previous iteration, where a Cox-ph model solution was used. This thesis seeks to find an algorithm that better predicts failures and as such, help better schedule maintenances, and better allocate specialized components, and human resources. This will also help understand how variations on context criteria, as terrain, wind speed affect the life cycle of components, and also choosing the technology that has the best life cycle.

1.4 Document structure

First there will be a revision of the literature, introducing first what is machine learning, Section 2.1, then introducing a methodology to solve the problem this thesis seeks to address, Section 2.2, then dealing with data collection and preparation, Section 2.3, then addressing the several types of algorithms in machine learning, Section 2.4, then an introduction to some concepts of reliability, Section 2.5.

In Section 3, there will be a description of the data provided for this thesis and the changes made to the data. In Section 3.1, 3.2 and 3.3, there will be a diagnose of the data, singling out

problems in the data. In Section 3.4, the actions taken to solve the problems with the data and adjusting the data to better work with the algorithms.

In Section 4, there will be a justification of what features were chosen to be fed to the data and why.

In Section 5, there will be an introduction to what are the goals of the algorithm, Section 5.1, and then what algorithm was chosen, Section 5.2, and how to address all the manual parameters of the algorithm, Section 5.3.

In Section 6, there will be a description of the optimization goals arbitrated, Section 6.1, and a description of the iterative training process, Section 6.2.

In Section 7, there will be a performance analysis on the optimization goals, Sections 7.1 and 7.2, then a result comparison with previously implemented technologies, Section 7.3.

Finally in Section 8, It will start with a description of this thesis goals and if they were achieved, Section 8.1, and a suggestion of future work to be done, Section 8.2.

Chapter 2

Literature review

In this chapter we will perform an introduction to the literature regarding the topic of machine learning and it's field of studies, combined with some auxiliary concepts of reliability and data analysis. It will take us from the data preparation, to the data transformation, and end with prediction algorithms, in search for a solution that fits the problem.

2.1 Machine learning vs statistical analysis

It is significant to expand on the difference between machine learning and statistical analysis, due do the fact that this thesis, was defined as an alternative approach against a traditional statistical predictor model, the Cox proportional hazards model, which had already been applied to the problem at hand.

First, in anything one wishes to know, it is interesting to start with the motivation, the question each method seeks to answer.

- Machine Learning:"How can we build machines that solve problems, and which problems are inherently tractable/intractable?"(Mitchell,2006 [14])
- Statistics:"What can be inferred from data plus a set of modeling assumptions, with what reliability?"(Mitchell,2006 [14])

So what can be inferred from the motivations is that they have very similar goals but very different perspectives. Namely, machine learning has less issue with working with more of "black-box" models. From, this a problem arises, as (Kantardzic 2011 [1]) points out "data-mining models should help in decision making. Hence, such models need to be interpretable in order to be useful because humans are not likely to base their decisions on complex "black-box" models. Note that the goals of accuracy of the model and accuracy of its interpretation are somewhat contradictory". So, it can be observed that the differences are subtle but important, because statistical analysis has a much bigger emphasis on the "readability" of the process, whereas machine learning algorithms are concerned with the readability of the results, but not necessarily of the process, being to the user, somewhat of "black box models" even though they have sound theoretical algorithms and

great accuracy. This shows that in order to be able to have a full understanding of the model and in order to be easily understood and justifiable, there has to be a process and a sound methodology.

2.2 The machine learning process

When tackling a problem using machine learning, (Marsland 2014 [4]) offers a process and methodology to the approach, the "machine learning process", and defines it as a process by which machine learning algorithms can be selected, applied, and evaluated. The approach is divided in 6 steps, *Data Colection and Preparation*, *Feature Selection*, *Algorithm Choice*, *Parameter and Model Selection*, *Training* and *Evaluation*.

Other approaches do exist, but even though they are different in terminology they are somewhat similar in substance, for example in Kantardzic 2011 [1], "the data-mining process" is described as: "state the problem", "collect the data", "perform preprocessing", "estimate the model (mine the data)", "interpret the model and draw the conclusions".

This process helps polish the data, choose the strongest algorithm and as such have an accurate and intelligible answer to the problem.

2.3 Data collection and preparation

When being given raw data, or unprocessed data, some problems arise, in fact as ,Kantardzic 2011 [1], points out, the raw datasets are often large and have the potential for being messy, so in fact there is often great entropy in the data, such as missing values, distortions, mis-recordings and inadequate sampling. This could have a major impact in the algorithm choice, because many algorithms are sensitive to missing values, so it may require an alternative algorithm or pre-processing the data.

In Pyle 1999 [15] the objective of preparing data is so that the "data miner" is able to end up with a data set that produces better and more intelligible models. Nevertheless it is important to perform this process done without disrupting the natural order of the data, risking compromising the end goal.

The data collection and preparation are a set of steps, such as normalizing the data and converting non numerical data, that aim at shaping the data into something the algorithm can better process and optimize.

2.3.1 Handling non-numeric variables

A key aspect and hurdle of any dataset is non-numeric data, as many processing algorithms only operate on numeric data. Therefore a transformation on the data is needed, converting non-numeric data to a numeric representation.

Table 2.1: Cities example of a non-numeric to numeric transformation, using the One-of-n remapping

City	City:Porto	City:London	City:Edinburgh
Porto	1	0	0
London	0	1	0
Edinburgh	0	0	1
Porto	1	0	0
London	0	1	0

2.3.1.1 One-of-n remapping

A one-of-n remapping consists on creating a binary-valued pseudo-variable for each non-numeric label value (Pyle 1999 [15]). In this process a new variable is created, that is "1" if the label is present, or "0" otherwise. If for example, there is a dataset with the names of three cities, Table 2.1, this will result in three new pseudo binary variables.

The advantage of this method is that the mean of the pseudo-variable, is proportional to the proportion of the label in the dataset. The disadvantages are, the large increase in dimensionality, that can hamper computationally the modeling and, in cases of low density, the prediction algorithms can find it difficult to distinguish from zero.

2.3.2 Outlier detection

According to Pyle 1999 [15], an outlier is a single, or very low frequency occurrence, that is far away from the bulk of the values of the variable, Figure 2.1, they can be the result of a mistake, or a result of the normal variability, and as Kantardzic 2011 [1] points out the data-mining analyst has to be very careful when applying an automatic elimination of outliers, because if the data is correct, it can result in the loss of important hidden information. An example of that, can be the algorithm detecting credit card fraud, what can be considered as an outlier may well be a case of credit card fraud and not a sampling error, therefore one cannot just dismiss every outlier as an error.

Nevertheless, often outliers are caused by measurement errors due to a device malfunction error, or an error reading data and can have serious effect in the machine learning algorithm. If the true value of the missing value is known one can replace it, but in most cases, such is unknown. If the true range is known, one can implicitly detect it as an outlier, whereas if such is not the case a threshold can be created, related to the distance to the mean, usually proportional to the standard deviation of the distribution of the variable. If a normally distributed random variable is assumed, a distance of two times the standard deviation covers 95% of the points, and a distance of three times the standard deviation covers 99% of the points, so it is up to the user to arbitrate what constitutes an outlier (Theodoridis & Koutroumbas 2008 [16]).

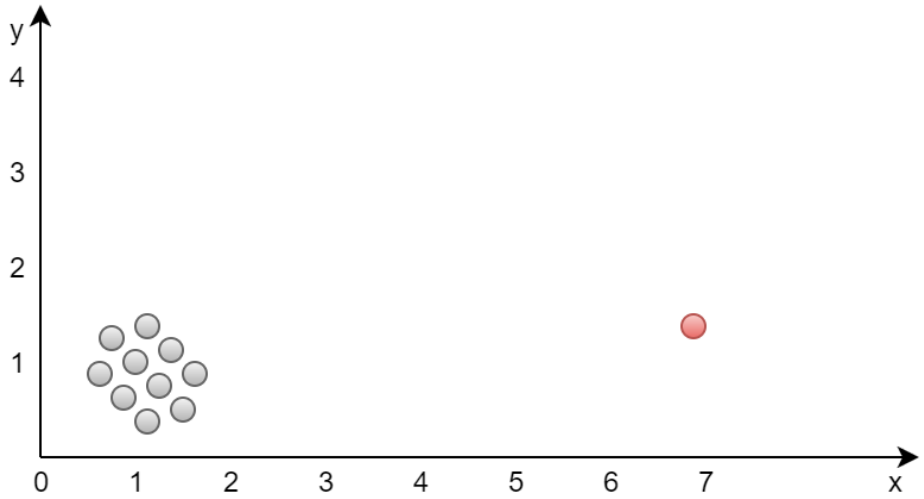


Figure 2.1: Two dimensional outlier example

As above mentioned, the models are sensitive to outliers, and in order to solve this predicament, two strategies can be used. If the number of outliers is very small, they can be discarded (Theodoridis & Koutroumbas 2008 [16]), however some models are unable to use observations where some of the variable values are missing and as such, the whole multi-dimensional observation needs to be discarded, which reduces the amount of information available for modeling and can hurt the overall accuracy of the model. The other alternative is replace the outlier or the missing value with an estimate of its true value. The most traditional techniques in dealing with missing data include replacing the missing values by zeros or with the unconditional mean or the conditional mean, even if that introduces noise into the samples, although if there are only a small number of outliers, that shouldn't have a big impact on the overall model accuracy.

2.3.3 Missing and empty variables

When dealing with missing variables two options arise. If the data missing are few, compared to the remaining of the dataset, one can choose to simply delete it, as it probably will not affect the dataset. If on the other hand it is deemed relevant or of great impact to the dataset, they can be replaced by a value. This new value has to be carefully chosen as to not skew the data, adding bias and distortion to the data (Pyle 1999 [15]).

Such considerations of what method to choose to estimate the new value are always a matter of a trade-off, between the impact this prediction has to the dataset, and the impact it has in the computational time. If the method is complex and the missing data is large, obviously it will take a long time to predict the values. On the other hand, simple methods may not be enough, or may affect the accuracy of the output prediction heavily down the road.

Now, two traditional methods are gonna be introduced, a mean estimator and multiple linear regression.

2.3.3.1 Mean estimators

A simple and trivial way to address the missing data is to replace it with a mean of the variable. This simple method has a low computational time and a simple implementation. The downside of using the mean of the data is that it highly disturbs the intrinsic variance of the data, which may be very relevant.

2.3.3.2 Multiple linear regressions

A multiple linear regression is an extension of a linear regression for multiple variables, more than two. It assumes the missing variable has a linear correlation to other variables, which many times is a reasonable assumption. The method tries to fit a linear equation to the observed data. This method suffers from a straightforward downside, which is an assumption that the variables are linearly correlated.

The model can be described as:

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + ... + \beta_p x_{pi} + \varepsilon_i, \qquad i = 1, 2, ..., n \tag{2.1}$$

Where y is value of the dependent variable, x are the independent variables, p are the number of explanatory variables, β correspond to the fitting parameters adjusting the linear equation and ε is called the residual and is the deviation of the predicted values, y_i, from their mean, μ_i.

All the linear and non linear methods described on Section 2.4, can also be applied to predict missing values.

2.3.4 Normalization

Normalizing, is turning features with dynamic ranges and scaling them to [0,1] ranges. This is important because features with large values may have a larger influence in the cost function than features with small values, which may not necessarily reflect their respective significance to the prediction algorithm (Theodoridis & Koutroumbas 2008 [16]), therefore having every feature on the same range solves that problem. Several techniques can be applied to do this transformation as for example a normalization using the linear scaling transform where $x1, ..., x_N$ are all the values in the sample, x_n is the original instance value, and x_{nN} is the normalized value (Pyle 1999 [15]).It can be seen in equation (2.5), the linear scaling transform.

$$x_{n_N} = \frac{x_n - min(x_1 ... X_N)}{max(x_1 ... x_N) - min(x_1 ... x_N)} \tag{2.2}$$

From Normalization, two problems may arise, the problem of the range of the sample versus the true range, which may affect variance or minimum and maximum of the population, that could result in the appearance of values outside the [0,1] range, the second problem are outliers

that artificially amplify the scale. The first may be solved by having a sufficiently representative sample and the second can be solved by dealing with the outliers as explained in Section 2.3.1.

2.3.5 Standardization

Standardizing, is transforming the data in order that it becomes zero mean and unit variance. This is done using estimates of mean and standard deviation, calculated from the sample data, where x_n is the original instance value, $\bar{x}$ is the sample mean, σ is the sample standard deviation, and x_{n_S} is the standardized value. In equation (2.6), it can be seen the standardization formula.

$$\bar{x_k} = \frac{1}{N}\sum_{i=1}^{N} x_{ik}, k = 1, 2, ..., l \tag{2.3}$$

$$\sigma_k^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_{ik} - \bar{x_i}k)^2 \tag{2.4}$$

$$x_{n_S} = \frac{x_n - \bar{x}}{\sigma} \tag{2.5}$$

2.3.6 Feature selection & extraction

A feature, is the equivalent to a variable in a machine learning setting. The decision of what variables to consider and discard, is one of the fundamental steps in the data analysis process, largely conditioning the success of any subsequent statistics or machine learning challenge, as such this step must be treated carefully (Guyon & Elisseeff 2006 [12]). The objective of feature selection is mainly, to select relevant and informative features, even though it may have other purposes, as *general data reduction*, to decrease the amount of data storage required and increase the speed of the algorithm, *feature set reduction*, save resources in the next round of data collection or during utilization, *performance improvement*, to gain predictive accuracy and in order to get a better *data understanding*.

2.3.6.1 Feature selection techniques

There can be discerned three major approaches, Table 2.2.

- **Filters**- Algorithms that carry out the pre-processing independently of the induction algorithm, it is used later.
  - Univariate- Algorithms that analyze the variable individually, therefore ignoring dependencies, even though they are fast and scalable (Bólon-Canedo et all 2012 [11]),

specially when the number of features is large and the number of available training examples comparatively small (Guyon & Elisseeff 2006 [12]) .

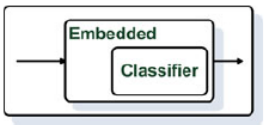
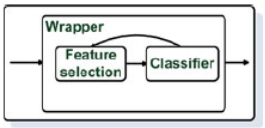
- Multivariate- Algorithms that are able to model feature dependencies, but are slower and less scalable than univariate methods (Bólon-Canedo et all 2012 [11]).

- **Wrappers**- A process that optimizes the prediction as part of the selection process.
- **Embedded** - Are utilized in the training process and are mostly specific to a given machine learning algorithm.

Other types of classification methods can be:

- Individual Evaluation / Feature Ranking- Assigns weights to individual features according to their degrees of relevance. It is incapable of removing redundant features, because they will have near weights.
- Subset Evaluation- It creates a subset of candidate features, then each candidate subset is evaluated and compared to the previous best.

Table 2.2: Summary of the feature selection techniques, characteristics, advantages, disadvantages and examples of algorithms [11].

Method	Advantages	Disadvantages	Examples
Filter 	Independence of the classifier Lower computational cost than wrappers Fast Good generalization ability	No interaction with the classifier	Consistency-based CFS INTERACT ReliefF $\mathcal{M}_d$ Information Gain mRMR
Embedded 	Interaction with the classifier Lower computational cost than wrappers Captures feature dependencies	Classifier-dependent selection	FS-Percepton SVM-RFE
Wrapper 	Interaction with the classifier Captures feature dependencies	Computationally expensive Risk of overfitting Classifier-dependent selection	Wrapper-C4.5 Wrapper SVM

2.3.6.2 ReliefF

Relief, is a filter algorithm, first proposed by Kira and Rendell,1992 [17]. The original method had the downsides of not being able to deal with incomplete data and being limited to two-class problems (Marko Robnik-Sikonja & Igor Kononenko 2003 [13]). ReliefF is an extension to the

Table 2.3: Frequently used feature selection methods.:RLSQ=regularized least square; RFE=recursive feature elimination; SVM=support vector machine; OBD=optimum brain damage; ARD=automatic relevance determination; RF=random forest. m is the number of training examples and n is the number of features [12].

Feature selection	Matching classifier	Computational complexity	Comments
Pearson (Eq. 1)	Naïve bayes	nm	Feature ranking filter. Linear univariate. Makes independence assumptions between features. Low computational and statistical complexity.
Relief (Eq. 2)	Nearest neighbors	nm^2	Feature ranking filter. Non-linear multivariate. Statistical complexity monitored by the number of neighbors.
Gram-Schmidt (Sec. 3.4)	linear RLSQ	fnm	Forward selection, stopped at f features. Linear multivariate. The statistical complexity of RLSQ monitored by the regularization parameter or "ridge".
RFE-SVM (Sec. 3.4)	SVM	$\max(n,m)m^2$	Backward elimination. Multivariate, linear or non-linear. Statistical complexity monitored by kernel choice and "soft-margin" constraints.
OBD/ARD	Neural Nets	$min(n,m)nmh$	Backward elimination. Non-linear multivariate. Statistical complexity monitored by the number h of hidden units and the regularization parameter or"weight decay".
RF	RF	$t\sqrt{n}\ m\log m$	Ensemble of t tree classifiers, each preforming forward selection. Non-linear multivariate.

Relief algorithm, proposed by Kononenko 1994 [18], that improves on solving the limitations of the original Relief, being able to deal with multi-class problems, more robust and able to deal with incomplete and noisy data.

It can be seen in Table 2.4 the algorithm for the ReliefF method. First it randomly selects an instance R_i from the data, then locates it's k nearest neighbors, from the same class, named nearest hits H_j, also it searches for k nearest neighbors from each of the different classes, named nearest misses $M_j(C)$. Then it updates the quality estimation $W[A]$ for all attributes A, in relation to the values for R_i, and hits H_j and misses $M_j(C)$. In the update formula, it's averaged the contribution for all the hits, H_j, and misses, $M_j(C)$. The contribution for each class of the misses is weighted with the prior probability of that class P(C). In order to consider the class of hits it's required to divide each probability weight with a factor 1-P(class(R_i)), the whole process is repeated m times (Marko Robnik-Sikonja & Igor Kononenko 2003 [13]). The k selector of hits, defined by the user, usually 10, results in a more robust algorithm regarding noise (Kononenko 1994 [18]).

Feature extraction means transforming raw features into new features that are transformations

Algorithm ReliefF
Input: for each training instance a vector of attribute values and the class value
Output: the vector W of estimations of the qualities of attributes

1. set all weights $W[A] := 0.0$;
2. **for** i := 1 **to** m **do begin**
3. randomly select an instance R_i;
4. find k nearest hits H_j;
5. **for** each class $C \neq class(R_i)$ **do**
6. from class C find k nearest misses $M_j(C)$;
7. **for** $A := 1$ **to** a **do**
8. $W[A] := W[A] - \sum_{j=1}^{k} \text{diff}(A, R_i, H_j)/(m \cdot k) +$
9. $\sum_{C \neq class(R_i)} [\frac{P(C)}{1-P(class(R_i))} \sum_{j=1}^{k} \text{diff}(A, R_i, M_j(C))]/(m \cdot k)$;
10. **end**;

Table 2.4: RelieF pseudo-code [13]

of the original ones. They can result in a combination of features that are a transformation of the original feature, as for example, with the normalization or standardization it was referred of previously (Sections 2.3.2 and 2.3.3).

2.4 Models

In data mining, as (Kantardzic 2011 [1]) explains, "In practice, the two primary goals of data mining tend to be prediction and description. Prediction involves using some variables or fields in the data set to predict unknown or future values of other variables of interest. Description, on the other hand, focuses on finding patterns describing the data that can be interpreted by humans", hence there are two major challenges: to predict and to describe.

In predictions, a model of the system is created, which can be easily understood by thinking of a two dimensional function, where the objective would be, given a set of x and y, to discover the f(x) which relates both variables. Both this objectives are achieved by this *primary data mining tasks*(Kantardzic 2011 [1]):

1. **Classification** - discovery of a predictive learning function, that classifies a data item into one of several predefined classes.

2. **Regression** – discovery of a predictive learning function, which maps a data item to a real-value prediction variable.

3. **Clustering** – a common descriptive task in which one seeks to identify a finite set of categories or clusters to describe the data.

4. **Summarization** – an additional descriptive task that involves methods for finding a compact description for a set (or subset) of data.

5. **Dependency Modeling** – finding a local model that describes significant dependencies between variables or between the values of a feature in a data set or in a part of a data set.

6. **Change and Deviation Detection** – discovering the most significant changes in the data set.

Other ways one can analyze a problem are related to the available data, which are obviously related as well to the objective. There are three big groups:

1. **Supervised Learning** - A teacher assigns a category label or cost for each pattern in a training set, and tries to minimize the sum of the costs for these patterns (Duda et al. 2001 [19]). In supervised learning, the algorithm has access to a "teacher-fitness function", where the outputs for the training set are known (Kantardzic 2011 [1]). In Figure 2.2, one can see how the teacher provides the outputs, and the system, through the feedback error, creates a model.

2. **Unsupervised Learning** or Clustering - In this event, only samples with input values are given to a learning system, and there is no notion of the output during the learning process. Therefore, there is no teacher and the learner forms and evaluates the model on its own. The main aim of unsupervised learning is to discover "natural" structures in the input data (Kantardzic 2011 [1]). Here, the objective is not to find a model that relates inputs and outputs, but to discover correlations between inputs, forming a set number of "clusters". In Figure 2.3, it can be seen how in unsupervised learning the system has no access to outputs nor any support.

3. **Reinforcement learning** - In reinforcement learning or learning with a critic, no desired category signal is given, instead, there is only a binary yes or no feedback from a teacher, as displayed in Figure 2.4. "This is analogous to a critic who merely states that something is right or wrong, but does not say specifically how it is wrong."(Duda et al. 2001 [19]).

In this work, the aim is to use a model able to make predictions into the future based on past occurrences, a *predictive model*, predictive models could be see as learning a mapping from an input set of vector measurements x to a scalar output y. Therefore, the aim of a predictive model is to estimate (from a training set, teacher) a mapping or a function $y = f(x;?)$ that can predict

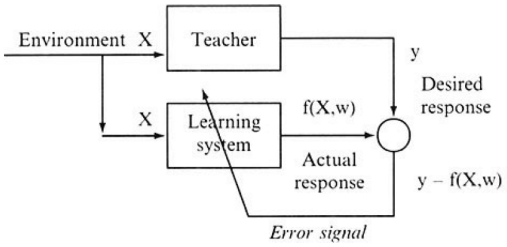


Figure 2.2: Supervised learning [1]

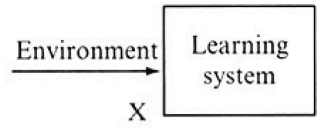


Figure 2.3: Unsupervised learning [1]

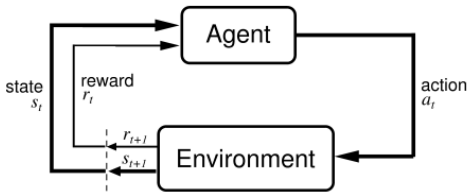


Figure 2.4: Reinforcement learning [2]

a value y given an input vector of measured values x and a set of estimated parameters ? for the model f (Hand et al. 2001 [20]).

There are two different types of predictive models, depending upon the response, if it's categorical, *classification*, or real-valued, *regression*.

2.4.1 Classification

In classification, there is a learning function that classifies a data item into one of several predefined classes, or in other words, it seeks to learn a mapping from a vector of measurements x to a categorical variable y. As shown in Figure 2.5, the training set, and in 2.6, a classification function, that separates the data in two classes "ball" and "star".

2.4.2 Regression

In regression, the end result of the learning process is a learning function, that maps a data item to a real-value prediction variable. Based on this function, it is possible to predict the value of a variable for each new sample (Kantardzic 2011 [1]). In Figure 2.7, one can look at the difference, whereas in classification the objective is to partition the space state in order to classify, in regression the objective is to create a model that approximates the result, and as such is able to generate predictions of the output according to a new input.

2.4.3 Cross validation

The cross validation is a model evaluation method, that helps estimate how good the model will behave when faced with new data. When a model is trained using a training set with known outputs, there are little guarantees that the model is indeed a model that generalizes well for real

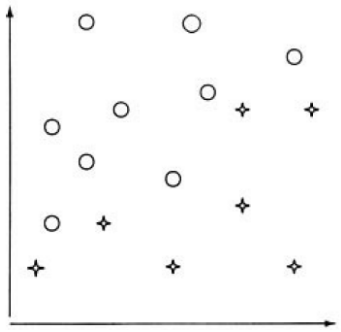


Figure 2.5: Classification training set [1]

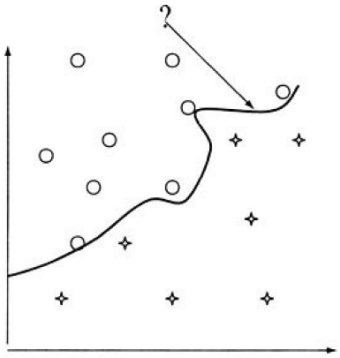


Figure 2.6: Classification prediction function [1]

data, as such it is required further validation to verify that the model works and an estimate of it is accuracy.

2.4.3.1 Hold-out method

The hold-out method, is the most common, the training data is broken into two new datasets, one is used to train and the other to test the model. The ratio of the split depends on the size of the dataset, if large typically 75% of the dataset is used for training and 25% for the test, if small 90% for training and 10% for test. The model is confronted with the test set and the cumulative mean absolute error, between the predicted outcome and the real outcome, is used to evaluate the model. This evaluation can have a high variance and is dependent on which datapoints end up in the train and test set, which can affect the evaluation [21].

2.4.3.2 K-fold cross validation

This method is an increment upon the hold-out method, it divides the training set into k subsets, and the hold-out methods is repeated a k number of times. At every iteration, one of the k subsets is used as the test and the others $k-1$ subsets are put together to form a training set. Then the average error across all k trials is computed. The advantage is that the data split is less relevant and every datapoint gets to be in the test set at least once. The disadvantage is that it has to be run k iterations, which means it requires k times more computationally than the hold-out method [1].

2.4.4 Neural networks

Neural networks start as an attempt to mirror biological systems, namely a neuron. This systems are important due to the fact that even though the computer has higher levels of speed computation, the human brain has interesting abilities. It can learn from experience, the performance does not degrade appreciably under damage, it is robust, and it performs parallel computations quite well.

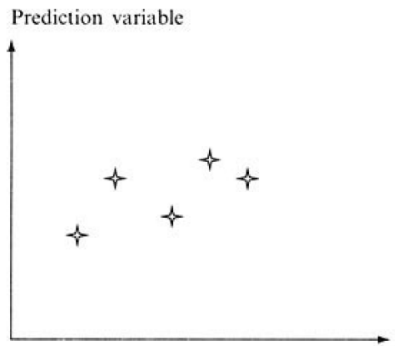


Figure 2.7: Regression training set [1]

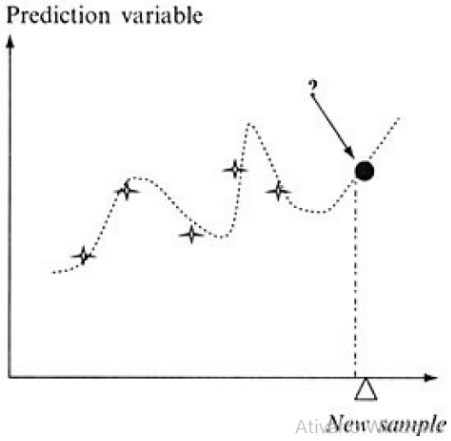


Figure 2.8: Regression prediction function [1]

2.4.4.1 Perceptron

The main concept, as (Marsland 2014 [4]) explains, "principal concept is plasticity: modifying the strength of synaptic connections between neurons, and creating new connections". The neuron can be mathematically described as Figure 2.9, as originally suggested by (McCulloch and Pitts 1943 [3]). This of course is not a true reflection on how a neuron works, but it is close enough and gives rise to interesting capabilities.

In Figure 2.9, one can observe how a neuron is constituted by a set of weighted inputs w_i, that corresponds to the synapses of the neurons, an adder that sums the input signals, equivalent to the membrane of the cell that collects electrical charge, and an an activation function, initially a threshold function, that decides whether the neuron fires for the current inputs. This neuron is able to give a binary answer, for instance, if above the threshold and beneath it.

Simplifying, the inputs x_i are multiplied by the weights w_i, and the neurons sum their values. If this sum is greater than the threshold then the neuron fires, otherwise it does not. (Marsland 2014 [4])

$$h = \sum_{i=1}^{m} w_i x_i \tag{2.6}$$

Equation (2.7), is the activation function, where w_i are the weighted inputs x_i the inputs, m the number of inputs.

From what it was described, this model is very simple, and very limited as it can only give a binary response, and lacks the learning ability. In neural networks, the model is expanded and treated in a parallel nature.

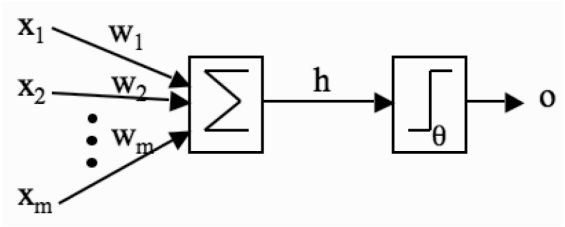


Figure 2.9: A picture of McCulloch and Pitts' [3] mathematical model of a neuron.

2.4.4.2 Perceptron network

The perceptron network, is an aggregation of McCulloch and Pitts neurons, together with a set of inputs and weights to correlate the inputs to the neurons (Marsland 2014). In Figure 2.10, one can notice how instead of a single neuron, a perceptron works with several independent neurons in parallel, and works out whether or not to fire, by multiplying together its own weights, and the input, adding them together, and comparing the result to its own threshold, regardless of what the other neurons are doing. Even the weights that go into each neuron can be different, so the only thing they share are the inputs. Therefore, every neuron computes all of the inputs to the network (Marsland 2014 [4]). In this instance the number of inputs is the same as the number of neurons, but this is not a required condition of the perceptron network. It is the aim of the perceptron to, given a training set, to predict target outputs.

The neuron will now apply the rule, and describe a firing pattern, based in if the neuron fired or not. This pattern will typically be of "0"s and "1"s. If the output pattern is equal to the target, t, this is the final model, with weights correctly adjusted, if not, a new iteration is need where the weights are updated. So it requires an error function, that is used to update the weights, giving them more "weight", or less "weight" according to the deviation from the target objective. In order to do this, first it is computed the error, according to equation (2.8), being y_k, the predicted output, minus the t_k, the target, and correct output. In order to get around the fact of the disturbance of a negative input, the formulation, presented in equation (2.9), is used instead. The new value for the weight will be the old one, plus this delta, as shown in equation (2.10).

$$\delta_k = y_k - t_k \tag{2.7}$$

$$\Delta w_i k = \delta_k \times x_i \tag{2.8}$$

$$w_i(k+1) = w_i(k) + \Delta(k) \tag{2.9}$$

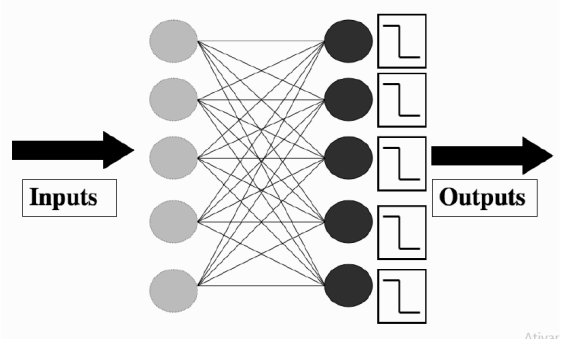


Figure 2.10: A perceptron network, consisting of a set of inputs (left) connected to a McCulloch and Pitts neurons using weighted connections [4]

Another question that must be addressed is how much the algorithm should change the weights, at what speed it should traverse the error surface, which is important in helping the network to not be stuck on a local minimum. In order to do that, the learning rule is adjusted. (Section 2.4.4.3), by multiplying it by a parameter called the learning rate, usually labeled η (Marslow 2014 [4]). So, the final rule for updating the weights will be:

$$\Delta w_i k = -\delta \times x_i \times \eta \tag{2.10}$$

The threshold function, is a discontinuous function, with discontinuous functions one cannot use derivatives, which is important to be able to analyze the error surface, and crutial ahead when talking about non-linear prediction. The question, is the importance of having in the neuron a threshold shape, so the "fire", "not fire", concept still holds. Mathematically there are a set of function, that resemble a continuous representation of the threshold, S-shaped functions, called sigmoid functions (Marslow 2014), Figure 2.12, equation (2.12).

The sigmoid has another convenient characteristic which is that it's derivative is relatively simple.

$$S = \frac{1}{1+e^{-t}} \tag{2.11}$$

2.4.4.3 Learning Rate η

The learning rate, η, can be an important factor of the algorithm. if equal to 1 it generates an unstable network, that never "settles down", on the other hand, a small learning rate means that the weights need to see the inputs more often before they change significantly, so the network takes longer to learn. However, it will be more stable and resistant to noise (errors), and inaccuracies in the data (Marslow 2014 [4]). Thus there is a trade-off between a network that never stops, or one that may overstep solutions, in other words between a network that takes too long to learn, and

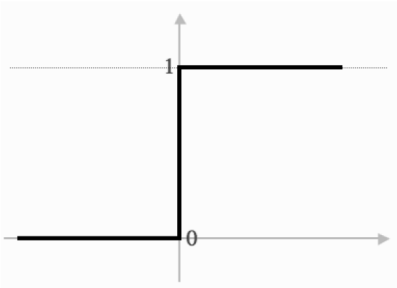


Figure 2.11: The threshold function, which has a discontinuity at 0. [4]

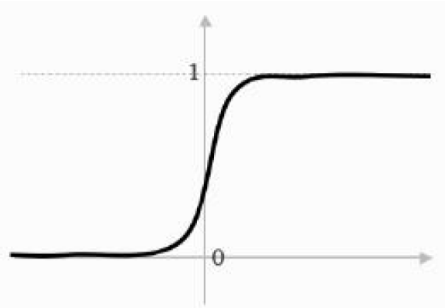


Figure 2.12: The sigmoid function, which is an S-shaped function. [4]

one that can be trapped in local minimums. Therefore, moderate learning rates are often applied, typically 0.1 <η< 0.4, depending upon how much error is expected in the inputs. This variable in practice is chosen by experience and it may also be time-varying, getting smaller as the algorithm progresses. In Figure 2.13 and 2.14, it can be seen how it may vary on an error surface.

The neurons all have a θ threshold, but the threshold should be adjustable, so that it can be changed when the neuron fires. So a bias is inserted, namely a $w_0 j$, which can be any non-zero value.

2.4.4.4 The Perceptron Network Algorithm

This is the Perceptron Network Algorithm according to (Marslow 2014 [4]).

- **Initization**
  - set all of the weights wij to small (positive and negative) random numbers
- **Training**
  - for T iterations or until all the outputs are correct:
    * for each input vector:
      – compute the activation of each neuron j using activation function g:

$$y_j = g(\sum_{i=0}^{m} W_{ij} x_i) = \begin{cases} 1 \, if \sum_{i=0}^{m} w_{ij} x_{ij} > 0 \\ 0 \, if \sum_{i=0}^{m} w_{ij} x_{ij} \leq 0 \end{cases} \tag{2.12}$$

      – update each of the weights individually using:

$$w_i(j+1) = w_i(j) + \Delta(j) \tag{2.13}$$

- **Recall**

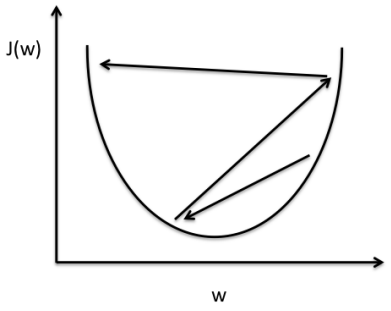


Figure 2.13: The problem of using a high learning rate, overshooting the global minimum [5].

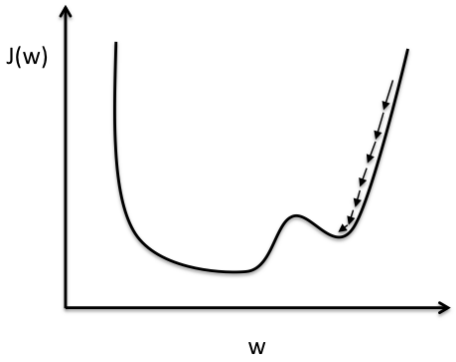


Figure 2.14: A small learning rate would take several iteration to arrive to the minimum, which may be problematic [5].

– compute the activation of each neuron j using:

$$y_j = g(\sum_{i=0}^{m} W_{ij}x_i) = \begin{cases} 1\, if\, \sum_{i=0}^{m} w_{ij}x_{ij} > 0 \\ 0\, if\, \sum_{i=0}^{m} w_{ij}x_{ij} \leq 0 \end{cases} \tag{2.14}$$

2.4.4.5 Multi-layer perceptron

With the perceptron, it was seen how with some neurons in parallel, it could achieve a function of weights, that could make a linear prediction, in other words, it could separate groups with straight lines, planes or hyperplanes. This approach is very limited, because the majority of complex problems are not linearly separable (Marslow 2014 [4]), therefore an alteration to the network scheme is needed in order to allow for non-linearity.

The solution for this problem is to add extra layers of neurons. This new layers are called "the hidden layers", they are called hidden due to the fact the correct activation functions, for this "middle" layers of neurons, are not known, Figure 2.15 and 2.16. So the multi-layer perceptron, will first compute the activations of the neurons, in the middle layer, and then will use those activations as the inputs to the single neuron at the output (Marslow 2014 [4]).

The model can be changed, in order to adjust to a non-linear activation function and to a new layer. This steps are called the forward phase or forward propagation.

In the first layer, a linear combination is applied as previously explained, equation (2.16). Where there are M linear combinations of D inputs, variables x_1,...., x_D, where $j = 1,...,M$ and (1) indicates the layer, w_{j0} are the biases and w_{ji} the weights, the quantities a_j are known as activations. (Bishop 2006 [6]).

$$a_j = \sum_{i=1}^{D} W_{ji}^{(1)}x_i + w_{j0}^{(1)} \tag{2.15}$$

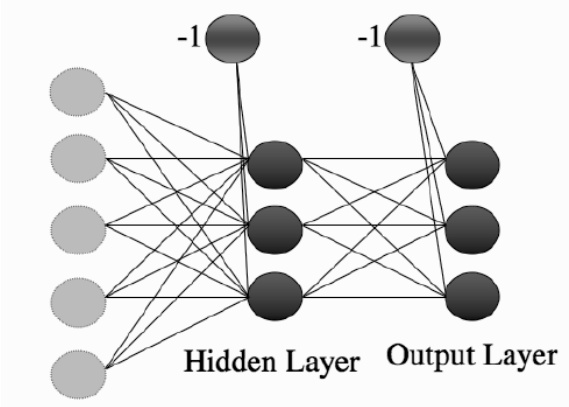


Figure 2.15: The multi-layer perceptron with a the input layer, the hidden layer and the output layer [4].

Now, each of them is then transformed using a differentiable, nonlinear activation function h(·), equation (2.17), , in this case the sigmoid function. Outputs of the basis function are called hidden units.

$$z_j = h(a_j) \tag{2.16}$$

The activation function is linearly recombined in order to give "output unit activations" (Bishop 2006 [6]), equation (2.18). Where K is the total number of outputs, $k = 1,...,K$, and $w_{k0}^{(2)}$ the bias in the second layer.

$$a_k = \sum_{j=1}^{M} W_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{2.17}$$

In the final step, the output unit activations, are transformed using an appropriate activation function, to give a set of network outputs, y_k, in standard regression problems, the activation function $y_k = a_k$, for multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid, equation (2.9),"for multiclass problems, a softmax activation function of the form is used".

$$y_k = \sigma(a_k) \quad ,where\ \sigma\ is \quad equation\ \mathbf{2.12} \tag{2.18}$$

Now it is possible to see this whole process in an aggregated fashion, the overall network function, for sigmoidal output unit activation functions, equation (2.20).

$$y_k(x,w) = \sigma\left(\sum_{j=1}^{M} W_{kj}^{(2)} z_j h\left(\sum_{i=1}^{D} W_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{2.19}$$

Now a neural network architecture has been achieved, it is a nonlinear function, from a set of input variables, x_i, to a set of output variables, y_k, controlled by a vector, w, of adjustable parameters, or weights.

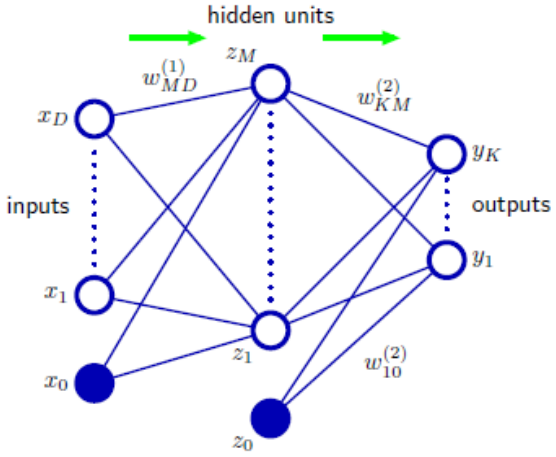


Figure 2.16: The two-layer perceptron, follows the same notation, of the equations previously introduced [6].

2.4.4.6 Error backpropagation

It is necessary, as it was seen for the simple perceptron, to have a training algorithm, "an iterative procedure for minimization of an error function, with adjustments to the weights being made in a sequence of steps" (Bishop 2006 [6]). The process can be broken down to two stages, first stage, the derivatives of the error function with respect to the weights must be evaluated, in the second stage, the derivatives are then used to compute the adjustments to be made to the weights. The simplest such technique, and the one originally considered by (Rumelhart et al. 1986 [22]), involves gradient descent. It was arbitrated, for this example, to use a sum-of-squares error function, multiplied by $\frac{1}{2}$ in order to simplify in the derivation, equation (2.21).

$$E(t,y) = \frac{1}{2}\sum_{k=1}^{N}(y_k - t_k)^2 \tag{2.20}$$

The gradient for this error, $\nabla E(w)$, with respect to a weigth, w_{ji}, is given by equation (2.22).

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \tag{2.21}$$

In order to proceed, it is is relevant to regard that, E_n, the error, depends on the weight, w_{ji}, only through the summed input, a_j, to unit ,j. This allows to apply the chain rule for partial derivatives, equation (2.23).

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{2.22}$$

Using an auxiliary variable, δ_j, and, z_i, equations (2.24) and (2.25).

$$\delta_j = \frac{\partial E_n}{\partial a_j} \tag{2.23}$$

$$z_i = \frac{\partial a_j}{\partial w_{ji}} \tag{2.24}$$

Combining the previous equations.

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \tag{2.25}$$

This correlation, informs that the required derivative is obtained simply by multiplying the value of δ for the unit at the output end of the weight by the value of z for the unit at the input end of the weight, where z = 1 in the case of a bias. Thus, in order to evaluate the derivatives, one only needs to compute the value of δ_j for each hidden and output unit in the network, and then apply equation (2.24).

To discover δ_j, for the hidden units, the link as the output-unit activation function, using the chain rule for partial derivatives, and where the sum runs over all units k that each unit j sends connections, equation (2.27).

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{2.26}$$

This technique uses the fact that changes in the activation function, a_j, give rise to changes in the error function through a_k.

It can now be written the following backpropagation formula by aggregating the relations, equation (2.27).

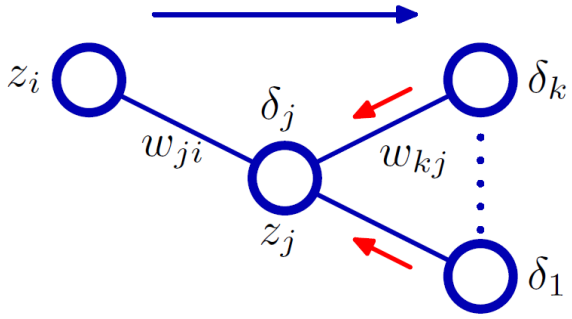


Figure 2.17: Demonstration of the calculation of δ_j for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information [6].

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta k \tag{2.27}$$

This formula explains that the δ, for a particular hidden unit, can be obtained by propagating the δ's backwards from units higher up in the network. Due to the fact that it is already known the values of the δ's for the output units, it follows that by recursively applying equation (2.28), it can evaluate the δ's for all of the hidden units in a feed-forward network, regardless of its topology.

2.4.4.7 Resilient back propagation

New methods have emerged that are improvements upon the back propagation algorithm, but use similar concepts. The algorithm works, in a way that when the partial derivative of the error surface, corresponding weight w_ij, changes sign, it indicates that the last update was too big and it has surpassed a local minimum, the updated value is then decreased by the factor η^-. Now, if the derivative maintains its sign, the update value is then slightly increased in order to speed up convergence in shallow regions. What happens is that if the weight w_ij is positive, the weight is decreased by the factor, if it is negative, it is increased by the factor. There is an exception, if the weight changes sign again, meaning the previous step was too large, the previous step is reverted and it ceases to adapt the weight (Martin Riedmiller et Heinrich Braun, 1993 [23]).

This algorithm is much faster than simple backpropagation and generally performs as well.

2.4.4.8 Multi-layer perceptron algorithm

The algorithm as proposed in (Marslow 2014 [4]).

- Initialization
  - initialize all weights to small (positive and negative) random values

- Training
  - repeat:

    *for each input vector:

    * Forwards phase:
      - compute the activation of each neuron j in the hidden layers using:

$$h_\zeta = \sum_{i=0}^{L} x_i v_{i\zeta} \tag{2.28}$$

$$a_\zeta = g(h_\zeta) = \frac{1}{1+e^{-\beta h_\zeta}} \tag{2.29}$$

      - work through the network until you get to the output layer neurons, which have activations

$$h_k = \sum_j a_j w_{jk} \tag{2.30}$$

$$y_k = g(h_k) = \frac{1}{1+e^{-\beta h_k}} \tag{2.31}$$

    * Backwards Phase
      - compute the error at the output using:

$$\delta_o(K) = (y_k - t_k) y_k (1-y_k) \tag{2.32}$$

      - compute the error in the hidden layer using:

$$\delta_h(\zeta) = a_\zeta (1-a_\zeta) \sum_{k=1}^{N} w_\zeta \delta_o(K) \tag{2.33}$$

      - update the output layer weights using:

$$w_{\zeta k} \leftarrow w_{\zeta k} - \eta \delta_o(K) a_\zeta^{hidden} \tag{2.34}$$

      - update the hidden layer weights using:

$$v_l \leftarrow v_l - \eta \delta_h(K) x_l \tag{2.35}$$

*(if using sequential updating) randomize the order of the input vectors so that you don't trains in exactly the same order each iteration.

∗ until learning stops

- Recall
  - use the Forward phase in the training section above

One must consider the problem of overfitting, which is when the network looses the ability to generalize and instead is just memorizing the training set. There are several ways to address this issue that is very present in neural networks, namely cross-validation, Section 2.4.3.

2.4.5 Recurrent neural networks

Another distinction can be made between networks based on their architecture connectivity. If a network has one or more cycles, if it is possible to trace a path to the unit back to itself, then that is called a *Recurrent Network*. A *Non Recurrent Network*, as the ones referred previously, are based on a feed forward idea, where there is a continuum between the inputs and the outputs, which is to say the networks follows a function that relates the inputs to the outputs, (Jordan, M. I. ,1986 [7]).

The recurrent network, not only depends on the input that it receives but also on the state of the network at a previous time step. For example, a simple network as shown in Figure 2.18, if μ is the value of the recurrent weight, and assuming the units are linear, the activation of the output unit at time t is given by equation (2.37).

$$x_2(t) = \mu x_2(t-1) + W_{21} x_1(t) \tag{2.36}$$

This shows how in this architecture the output depends not only on the input, but also on the previous state.

As explained, this type of architecture is interesting when the previous state of the network is relevant.

2.4.5.1 Jordan architecture

In order to create bigger networks, capable of non-linearity and multiple inputs and outputs, new architectures where created. One of the more common and simple are the Elman and Jordan networks. In Figure 2.19, the difference between a feed forward network and a Jordan Network.

In the Jordan architecture, the feedback connection is from the output layer to the recurrent input unit, with a a weight of 1. So at time t, the networks receives the weight from the recurrent

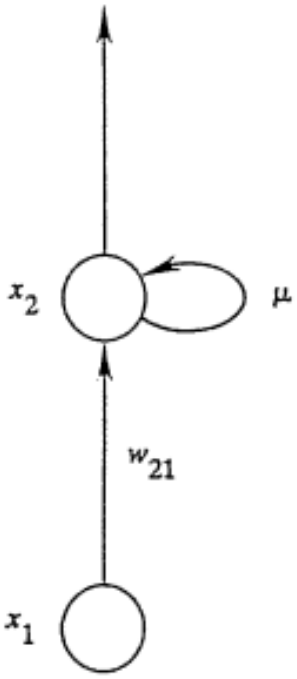


Figure 2.18: Example of a recurrent neural network with two units [7].

unit as an input, that comes form $t-1$, (Karunanithi et all,1992 [8]). Here describes the example for one input, but for n inputs the network requires an equal number of n recurrent input and output units.

Regarding Training of the recurrent network, it's now transparent that the network will have two errors, the generalization error, which was explained for the non recurrent networks and the prediction training, the value of the input variable i_t at time t is associated with the real value of the output variable at time $t+1$. The prediction error trains the network to predict the outputs of the next time step (Karunanithi et all,1992 [8]).

2.4.6 Support vector machines SVM

This is a model that was introduced by Vapnik in 1992 [24], and "provides very impressive classification performance on reasonably sized datasets."(Marslow 2014 [4]). Even though SVMs do not work well on extremely large datasets, due to being computationally expensive, this is still very interesting model.

2.4.6.1 Linear support vector machines

It is pertinent to, as it was done with neural networks, to start by explaining a simple linear model and then generalize to non-linear models.

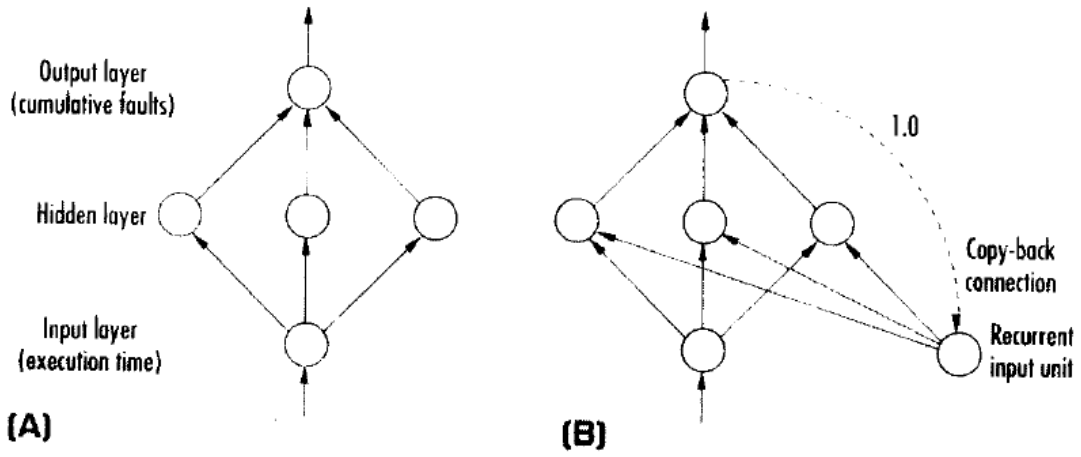


Figure 2.19: (A) Standard feed forward network and (B) a Jordan network [8].

In Figure 2.20, one can analyze three different classifications, and all are "correct", in the sense that all of them are able to classify and separate without error. Is any of the lines better? Which one was the best classifier? This introduces the concept of "optimal separation", the better line as it can be empirical ascertained, is the one that is furthest from the data points, and still is able to separate them. In this case the middle graph of Figure 2.21, because it introduces slack, if the line is close to a data point, the probability of misclassification is higher.

This notion is called the margin, and is defined by the smallest distance between the decision boundary and any of the samples. In the class of support vector machines the decision boundary is chosen to be the one for which the margin is maximized. The classifier with the biggest margin, is called the maximum margin (linear) classifier (Marslow 2014 [4]), the datapoints in each class that lie closest to the classification line are called "support vectors".

It is now possible to arrive to two conclusions, the margin should be as large as possible, and second that the support vectors are the key datapoints, because they are the ones that are more likely misclassified. This second conclusions gives rise to an interesting feature of these algorithms, after training one can dispense all of the data except for the support vectors, and use them for classification, which is a useful saving in data storage (Marslow 2014 [4]), due to being the important points when calculating the margin.

When talking of the perceptron an equation of the type of equation (2.37) was used, where $\mathbf{x}$, is an input vector, b, is the bias, and $\mathbf{w}$ the weight vector, one will now apply that to SVM. It is also worth noting that in SVMs, $\mathbf{w}$ is normal to the hyperplane (Burges 1998 [9]), meaning it's perpendicular to the classifier line (Marslow 2014 [4]), as can be seen in Figure 2.22.

$$y = \mathbf{w} \cdot \mathbf{x} + b \tag{2.37}$$

If one considers equation (2.37) in regard to Figure 2.22, and adding not wanting any datapoint

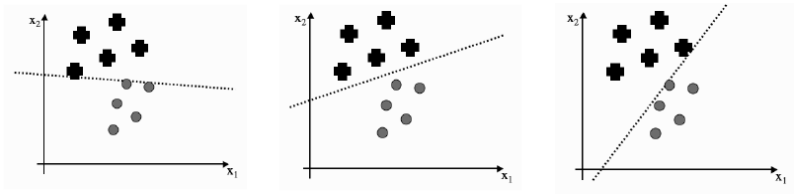


Figure 2.20: Three classification lines, all separate the sub-space [4].

in the grey area, one can say that for a margin M, any point x where $\mathbf{w}^T x + b \geq M$ is a plus, and any point where $\mathbf{w}^T \mathbf{x} + b \leq -M$ is a circle. The actual separating hyperplane is specified by $\mathbf{w}^T \mathbf{x} + b = M$ (Marslow 2014 [4]). Now what is needed is to traverse the perpendicular to the plus boundary line, a point x^+, until it's reached the circle boundary line, the point reached will be the closest and it can be called x^-.

The $\mathbf{w}$ is, as it was stated before, perpendicular to the classifier line, as such the perpendicular that was traversed was in fact the $\mathbf{w}$ vector. if it's now made w as a unit vector, $\mathbf{w}/\|w\|$, one can discover that the margin is $1/\|w\|$.

Due to the width of the margin being given by $1/\|w\|$, it's known that maximizing the width of the margin is the same as minimizing $\mathbf{w}^T \mathbf{w}$, or $\frac{1}{2}\mathbf{w}^T \mathbf{w}$, for convenience. It is now possible to write the SVM constraints. If it is assumed that the target, t, has two classes, for example "+1", "-1", the target can be multiplied for the output, equation (2.38).

$$t_i(\mathbf{w}^T \mathbf{x} + b) \geq 1, \quad \text{for all } i = 1, ..., n \tag{2.38}$$

The problem can now be put as in equation (2.39).

$$\text{minimize } \frac{1}{2}\mathbf{w}^T \mathbf{w} constrained by t_i(\mathbf{w}^T \mathbf{x} + b) \geq 1, \quad \text{for all } i = 1, ..., n \tag{2.39}$$

One could now use the method introduced in the neural network, the gradient descent, but it would be very slow and inefficient for the problem (Marslow 2014 [4]), a better way to solve this problem, is quadratic programming, which takes advantage of the fact that the problem described is quadratic and therefore convex, and has linear constraints.

A convex problem is one where taken any two points on the line and joining them with a straight line, would result in every point on the line being above the curve (Marslow 2014). Solving this quadratic problem has several advantages, the problem can be solved "directly and efficiently"(Marslow 2014 [4]) and there's a unique optimum (Marslow 2014 [4]).

2.4.6.2 Karush-Kuhn-Tucker KKT conditions

The KKT conditions are a set of conditions that are satisfied at the solution of a constrained optimization problem, with any kind of constraints, provided that the intersection of the set of

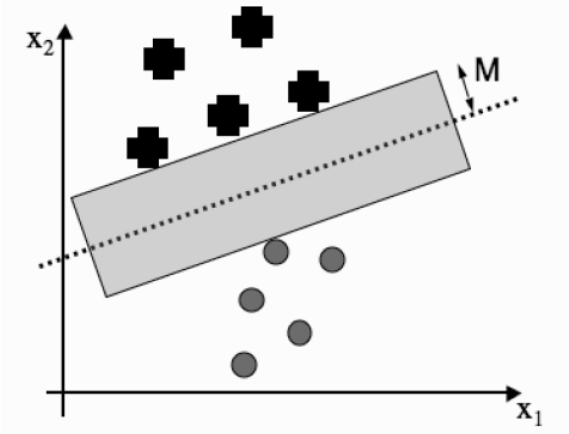


Figure 2.21: M is the margin, the grey box is the area where every line is a correct classifier for this set, but the dashed line, indicates the optimum solution that maximizes the margin.

feasible directions, with the set of descent directions coincides with the intersection of the set of feasible directions, for linearized constraints with the set of descent directions(Burges 1998 [9]).

Regarding the SVM problem the conditions are as stated in equations (2.40, 2.41 and 2.42), these are for all values of i from 1 to n, and where the λ_i denotes the optimal value of each parameter (Marslow 2014 [4]).

$$\lambda_i^*(1 - t_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*)) = 0 \tag{2.40}$$

$$1 - t_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*)) \leq 0 \tag{2.41}$$

$$\lambda_i^* \geq 0 \tag{2.42}$$

The λ_i are positive values known as Lagrange multipliers, which are a standard approach to solving equations with equality constraints.

The first of these conditions, equation (2.40), tells us that, if $\lambda_i \neq 0$ then $(1 - t_i(w^{*T}x_i + b^*)) = 0$. This is only true for the support vectors, so it's only needed to study them and not the whole set (Marslow 2014 [4]), in other words the support vectors are those in the active set of constraints, as such, for support vectors the inequalities are instead, equalities, turning it into a solvable Lagrange problem, equation (2.43).

$$\mathscr{L}(\mathbf{w}, b.\lambda) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n} \lambda_i(1 - t_i(\mathbf{w}^T\mathbf{x}_i + b)) \tag{2.43}$$

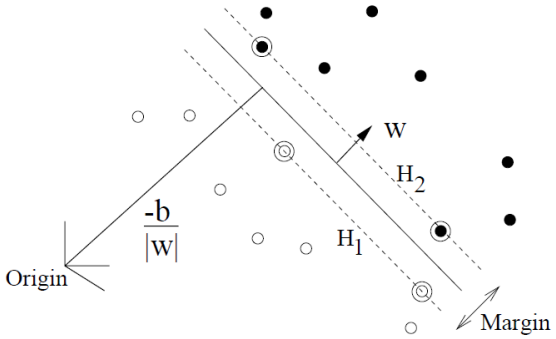


Figure 2.22: Linear separation of the hyperplanes H1 and H2, the unit vector w=$\mathbf{w}/\|w\|$ the support vectors are circled [9].

Differentiating with respect to $\mathbf{w}$ and b, equation 2.44, 2.45.

$$\nabla_{\mathbf{w}}\mathscr{L} = \mathbf{w} - \sum_{i=1}^{n} \lambda_i t_i \tag{2.44}$$

$$\frac{\partial \mathscr{L}}{b} = -\sum_{i=1}^{n} \lambda_i t_i \tag{2.45}$$

Now, equalizing the derivatives to zero, in order to search for the saddle points, maximum of the function, it can be arrived at equation 2.46.

$$w^* = \sum_{i=1}^{n} \lambda_i t_i x_i, \sum_{i=1}^{n} \lambda_i t_i x_i = 0 \tag{2.46}$$

Replacing the optimal values by the equations (2.44 and 2.45), it results in equation (2.46).

$$\mathscr{L}(\mathbf{w}, b.\lambda) = \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{n} \lambda_i t_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j t_i t_j x_i^T x_j \tag{2.47}$$

If it's considered the derivative, in respect to b, the $\sum_{i=1}^{n} \lambda_i t_i$ can be looked as 0. The equation (2.47), is known as the dual problem, and the objective is to maximize it with respect to the λ_i variables. The constraints are that $\lambda_i \geq 0$ for all i, and $\sum_{i=1}^{n} \lambda_i t_i = 0$ (Marslow 2014 [4]). In order to discover b^*, it's better to average it over the whole set of N_s support vectors, equation (2.49).

$$b^* = \frac{1}{N_s} \sum_{s_j} \left(t_j - \sum_{i=1}^{n} \lambda_i t_i x_i^T x_j \right) \tag{2.48}$$

For a new point $\mathbf{z}$, it can now be made a prediction, using equation (2.49).

$$\mathbf{w}^{*T}\mathbf{z}+b^{*}=\left(\sum_{i=1}^{n}\lambda_i t_i x_i\right)^{T}\mathbf{z}+b^{*} \tag{2.49}$$

Therefore in order to classify a new point, it's just need to compute the inner product between the new datapoint and the support vectors (Marslow 2014 [4]).

2.4.6.3 Non-linear support vector machines

Every step it was taken so far, implied that the dataset was linearly separable, which many times is not a reasonable assumption. In order to solve the problem of non-linearity, it's added a "positive slack variable", $\eta_i \geq 0$. This implies a change in the constraint, becoming instead equation (2.50), for correct classifications it was set as $\eta_i = 0$.

$$t_i(\mathbf{w}^T\mathbf{x}_i+b)\geq 1-\eta_i \tag{2.50}$$

Now, for an error to occur, the corresponding η_i must 1, so $\sum_i \eta_i$ is an upper bound on the number of training errors (Burges 1998 [9]). Another aspect that needs considering is the distinction, not just between right or wrong, but when comparing two wrong classifiers and deciding which one is better, in other words as (Marslow 2014 [4]) puts it "where one classifier makes a mistake by putting a point just on the wrong side of the line, and another puts the same point a long way onto the wrong side of the line. The first classifier is better than the second, because the mistake was not as serious, so this information should be included in the minimization criterion.". In order to do this, it's added a cost to the minimization problem, C, to the constrictions, where C is a parameter to be chosen by the user, a larger C corresponding to assigning a higher penalty to errors (Burges 1998 [9]), this parameter helps deciding how to update the weights, $w^T W + C\times$ (distance of misclassified points from the correct boundary line).

Now the problem is a soft-margin classifier, since now it's allowed a few mistakes.The derivation of the dual problem still holds, except that $0 \leq \lambda_i \leq C$, and the support vectors are now those vectors with $\lambda_i > 0$.(Marslow 2014 [4]). The new function that needs to be minimized is equation (2.51).

$$L(\mathbf{w},\varepsilon)=\mathbf{w}^T\mathbf{w}+C\sum_{i=1}^{n}\varepsilon_i \tag{2.51}$$

The KKT conditions also need to be rearranged to accommodate the new slack variables, equations (2.52, 2.53 and 2.54).

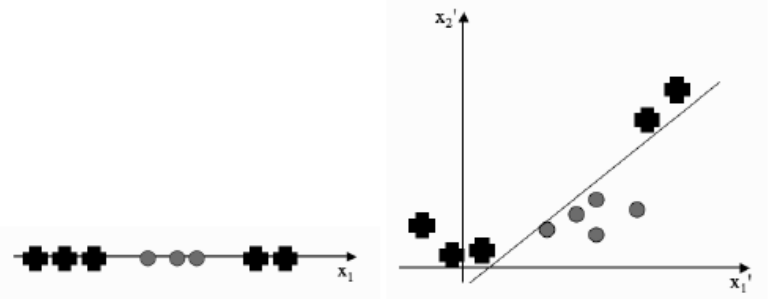


Figure 2.23: It can be seen how by transforming x_1 into, x_1^2 allows these two classes to be separated. Here the dimensions are changed from one to two, but the same is applicable to any dimension [4].

$$\lambda_i^*(1 - t_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*) - \eta_i) = 0 \tag{2.52}$$

$$(C - \lambda_i^*)\eta_i = 0 \tag{2.53}$$

$$\sum_{i=1}^{n} \lambda_i^* t_i = 0 \tag{2.54}$$

The second condition tells that, if $\lambda_i < C$, then $\eta_i = 0$, which means that these are the support vectors. If $\lambda_i = C$, then the first condition tells that if $\eta_i > 1$ then the classifier made a mistake (Marslow 2014 [4]).

The concept behind solving non-linear problems, is that if it could be used more dimensions, then it might be able to find a linear decision boundary that separates the classes, Figure 2.23. So all that it's needed to do is work out what extra dimensions that can be used (Marslow 2014 [4]), in order to do this the input vectors are transformed, using instead of x_i, a $\phi(x_i)$, equally z_i is changed for transformations, $\phi(z_i)$, of a greater dimension. So now the predictor function is, equation (2.55).

$$w^T\mathbf{x} + b = \left(\sum_{i=1}^{n} \lambda_i t_i \phi(\mathbf{x}_i)\right)^T \phi(\mathbf{z}) + b \tag{2.55}$$

A problem that wasn't solved is the computationally expensive method of multiplying $\phi(x_i)^T 1 \phi(x_j)$. For example, when confronted with the three dimensional problem, equation (2.56).

$$\phi(\mathbf{x})^{\mathbf{T}}\phi(\mathbf{y}) = 1 + 2\sum_{i=1}^{d} x_i y_i + \sum_{i=1}^{d} x_i^2 y_i^2 + 2\sum_{i,j=1;i<j}^{d} x_i x_j y_i y_j \tag{2.56}$$

This can be factorized to $(1+\mathbf{x}^{\mathbf{T}}\mathbf{y})^2$, "The dot product here is in the original space, so it only requires d multiplications, which is obviously much better — this part of the algorithm has now been reduced from $O(d^2)$ to $O(d)$. The same thing holds true for the polynomials of any degree s that we are making here, where the cost of the naïve algorithm is $O(d^s)$."(Marslow 2014 [4]), now all the dot products are replaced with the computation of a kernel matrix, $\mathbf{K}$, that is made from the dot product of the original vectors, that is linear in cost (Marslow 2014 [4]) this is "know as the kernel trick". This shows that $\phi(.)$ is not needed, if we know kernel. The kernel function can be any symmetric function that is positive definite (Marslow 2014 [4]). The most common used, according to (Marslow 2014 [4]) are:

- polynomials up to some degree s in the elements x_k of the input vectors with kernel:

$$\mathbf{K}(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}^T\mathbf{y})^s \tag{2.57}$$

  For $s = 1$ this gives a linear kernel.

- sigmoid functions of the $x_k s$ with parameters K and δ, and kernel:

$$\mathbf{K}(\mathbf{x},\mathbf{y}) = tanh(K\mathbf{x}^T\mathbf{y} - \delta) \tag{2.58}$$

- radial basis function expansions of the $x_k s$ with parameter σ and kernel:

$$\mathbf{K}(\mathbf{x},\mathbf{y}) = e^{(}\frac{-(\mathbf{x}-\mathbf{y})^2}{2\sigma^2}) \tag{2.59}$$

The task of selecting a kernel is not straightforward and "most people just experiment with different values and find one that works, using a validation set as we did for the MLP"(Marslow 2014 [4]).

In order to address how to do the computations for a testing set, one can use equation (2.56) and replace the computations of $\phi(\mathbf{x_i})^{\mathbf{T}}\phi(\mathbf{x_j})$, using the kernel as previously showed.

$$\mathbf{w} = \sum_{i\, where\, \lambda_i > 0} \lambda_i t_i \phi(\mathbf{x}_i) \tag{2.60}$$

A key factor of this method, is the objective of optimizing $\mathbf{w}^T\mathbf{w}$, which tries to keep $\mathbf{w}$ small, that results in many of the parameters being kept near 0 (Marslow 2014 [4]), therefore the overfitting problem disappears and it can always reach a global minimum.

2.4.6.4 The support vector machine algorithm

The SVM algorithm according to (Marslow 2014 [4]).

- **Initialization**
  - For the specified kernel, and kernel parameters, compute the kernel of distances between the datapoints
    *the main work here is the computation $\mathbf{K} = \mathbf{XX^T}$
    *for the linear kernel, return $\mathbf{K}$, for the polynomial of degree d return $\frac{1}{\sigma}\mathbf{K}^d$
    *for the RBF kernel, compute $\mathbf{K}$ = exp(-(x-x')^2/2σ^2)
- **Training**
  - assemble the constraints set as matrices to solve:

$$\min_x \frac{1}{2}x^T t_i t_j \mathbf{K}\mathbf{x} + \mathbf{q}^T\mathbf{x} \text{subject to } \mathbf{G}\mathbf{x} \leq \mathbf{h}, \mathbf{A}\mathbf{x} = b \tag{2.61}$$

  - pass these matrices to the solver
  - identify the support vectors as those that are within some specified distance of the closest point and dispose of the rest of the training data
  - compute b^* using equation
- **Classification**
  - for the given test data $\mathbf{z}$, use the support vectors to classify the data for the relevant kernel using:
    *compute the inner product of the test data and the support vectors
    *perform the classification as $\sum_{i=1}^{n} \lambda_i t_i \mathbf{K}(x_i, \mathbf{z}) + b^*$, returning either the label (hard classification) or the value (soft classification)

2.4.6.5 Support vector machine regression

The SVM was introduced as a classification solving algorithm, but with some alterations, it can be used as a regression algorithm as well. In order to do this, it's transformed using the least-squares error function (Marslow 2014 [4]), equation (2.62), and changing it by adding what is known as an insensitive error function, (E_ε), that gives 0, if the difference between the target and output is

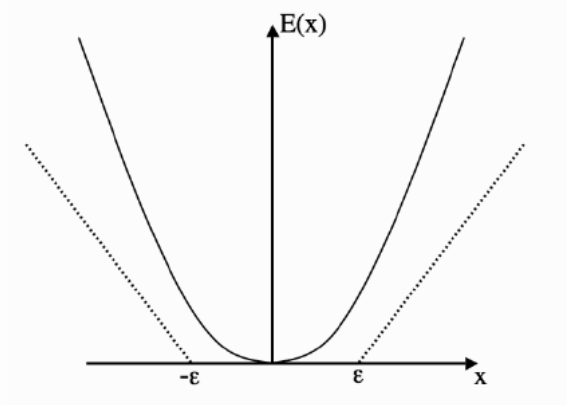


Figure 2.24: The ε-insensitive error function is zero for any error bellow ε [4].

less than ε, and subtracts ε in any other case for consistency. The rationale for this is that the goal is a small number of support vectors, so only points that are not well predicted are interesting (Marslow 2014 [4]), Figure 2.24.

$$\frac{1}{2}\sum_{i=1}^{N}(t_i - y_i)^2 + \frac{1}{2}\lambda\|\mathbf{w}\|^2 \tag{2.62}$$

In SVM regression, "we want the predictions to be inside the tube of radius ε that surrounds the correct line."(Marslow 2014 [4]), as with classification SVM, it needs to allow for errors, so again slack variables are introduced for each datapoint ,ε_i for datapoint i, with their constraints and follow the same procedure of introducing Lagrange multipliers, transferring to the dual problem, using a kernel function and solving the problem with a quadratic solver (Marslow 2014 [4]). In order to do predictions, one just needs to apply equation (2.63), where μ_i and λ_i, are two sets of constraint variables.

$$f(\mathbf{z}) = \sum_{i=1}^{n}(\mu_i - \lambda_i K(x_i, z) + b) \tag{2.63}$$

2.4.7 Decision trees

A tree-based model is a model that works by separating the input space into cuboid regions, whose edges align with the axes, and then corresponding a simple model to each region (Bishop 2006 [6]). It works as a sequential top to down binary decision, according to a decision criteria.

2.4.7.1 Binary tree

A binary tree is an algorithm of common use throughout the whole of computer science. It is computational cost is low, $O(log(N))$, where N is the number of datapoints (Marslow 2014 [4]).

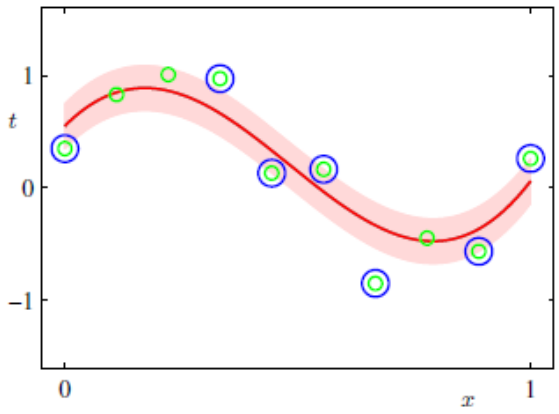


Figure 2.25: SVM for regression applied to the sinusoidal synthetic data set using Gaussian kernels. The predicted regression curve is shown by the red line, and the ε insensitive tube corresponds to the shaded region. Also, the data points are shown in green, and those with support vectors are indicated by blue circles (Bishop 2006 [6]).

A binary tree is a tree in which each node has only two children, so in essence the tree performs a binary decision, when descending from the root to the leaves.

The idea of a decision tree is to partition the classification into a set of decisions about each feature, starting at the "root" of the tree, the top, and traversing to the "leaves", down, where it receives the classification decision. In Figure 2.26 and 2.27, it can be seen how a binary tree can translate to a partition of the set. When considering an x, that we want to classify, it only needs to subject him to the top down tree conditions, and it will end up in a specific region classification.

A great advantage for trees in machine learning is the readability and comprehensibility, whereas a neural network is a hard method to explain and understand, being often read as "black box" method, decision trees can often be transformed into if rules that are easy to understand. Another set of advantages is that decision trees, aren't as strict in the data preparation as other algorithms, accepting non-numeric data.

2.4.7.2 CART classification and regression trees

In order to learn a tree model from a training set, it's necessary to create a structure of the tree, where it's defined an input variable at each node to form a split criterion, as well as a threshold parameter, θ_i, for the split, and finally the values of the predictive variable within each region (Bishop 2006 [6]). The structure selection of the tree cannot be an optimal choice, do the combinations associated with it, it would be computationally infeasible (Bishop 2006 [6]), therefore a greedy approach is selected.

A greedy optimization, starts with a single "root" node that encompasses all the input space, and then growing the tree by adding nodes one at a time. At each iteration there will be multiple candidate regions of the input space that can be split, corresponding to the increment of "leaves" to the tree. As such, for each iteration, there is a choice of what input variables to split and the value

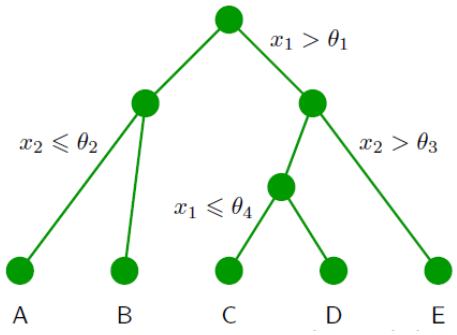


Figure 2.26: An example of a binary tree [6].

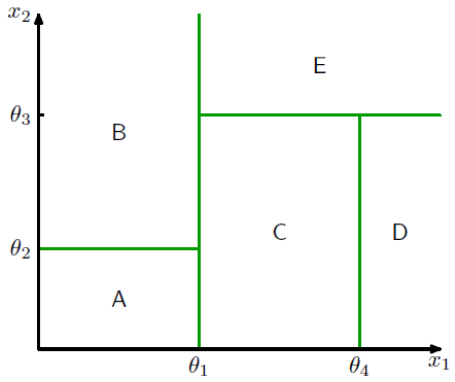


Figure 2.27: A partition of the set, using a binary tree [6].

of the threshold. The optimization of what region to split, and the choice of the input variable and threshold, can be done by exhaustive search, by noting that, for a given choice of a split variable and threshold, the optimal choice of predictive variable is given by the local average of the data (Bishop 2006 [6]). Repeating this process for all candidates, it can choose the one that minimizes or maximizes the decision criterion. The iteration can stop, by a pre defined stopping criteria, such as an error threshold, but it's common practice to grow the tree using the number of datapoint associated to the leaf nodes as a stopping criterion, and then prune back the tree (Bishop 2006 [6]).

When facing a regression problem, the objective is, for example, the minimization of the sum-of-squares error, as done in other methods. Now translating this mathematically, it can be described as equation (2.64).

$$y_\tau = \frac{1}{N_\tau} \sum_{x_n \in R_\tau} t_n \tag{2.64}$$

Where, τ is the leaf node correspondent to the input space, R_τ, having N_t datapoints. The corresponding minimization criterion for the residual sum-of-squares is, equation (2.65).

$$Q_\tau(T) = \sum_{x_n \in \aleph_\tau} (t_n - y_\tau)^2 \tag{2.65}$$

Being T the total number of leaf nodes. Now the pruning criterion can be described as, equation (2.66).

$$C(T) = \sum_{\tau=1}^{|T|} Q_\tau(T) + \lambda |T| \tag{2.66}$$

The pruning is based on a criterion that balances residual error against a measurement of the model complexity (Bishop 2006 [6]). The regularization parameter λ determines the balance

between the residual sum-of-squares error and the complexity of the model as measured by the number $|T|$ of leaf nodes, and its value is chosen by cross-validation (Bishop 2006 [6]).

The above example was an approach to regression using decision trees. When faced with a classification problem, the sum-of-squares is no longer an appropriate measure of performance.

Two commonly chosen measures are the cross entropy, equation (2.67), and the Gini index, equation (2.68).

$$Q_\tau(T) = \sum_{k=1}^{K} p_{\tau k} \ln p_{\tau k} \tag{2.67}$$

$$Q_\tau(T) = \sum_{k=1}^{K} p_{\tau k}(1 - p_{\tau k}) \tag{2.68}$$

Both these measures are zero for $p_{\tau k} = 0$ and $p_{\tau k} = 1$ and have a maximum at $p_{\tau k} = 0.5$. They bias towards the formation of regions with a high proportion of data points attributed to one class. They are both differentiable, so better suited for gradient based optimization methods and more sensitive to node probabilities (Bishop 2006 [6]).

The decision trees suffer from some disadvantages, they are normally less accurate than other methods, they are based in greedy algorithms, therefore they can arrive at non optimal results, they are not very robust (Hastie et al. 2001 [25]), a different value can result in a very different tree and another problem is that it splits in accordance to the axis which can be very suboptimal (Bishop 2006 [6]).

2.4.7.3 Random forests

Random Forests is an ensemble method, which means it results from a combination of models. The idea behind the ensemble methods is that a great number of models that arbitrate slightly different outputs, aggregated, are better than just one model. Several ensemble methods exist, but in order to understand Random Forests, one must look at bagging or bootstrap aggregating.

First, a bootstrap sample is an original dataset, with the same size of the dataset, that results from a random sampling of the dataset. So, now instead of using a dataset and applying a model algorithm to the dataset, it is created n new different datasets and have n different models, that generate n different outputs, and then combining them, applying a majority vote or a mean.

In Random Forests, it's used the ensemble concept applied to decision trees, if one tree is good, then a forest, group of trees, should be better. Provided that the trees have enough variety between them (Marslow 2014 [4]).

So the method implies taking bootstrap samples from the dataset, and generating trees for each dataset, at each node, a random subset of features is given to the tree, and it can only choose a subset, instead of the whole set. This helps increasing the randomness and the speed of the algorithm, due to not having to consider all the features, and typically only a subset that is the

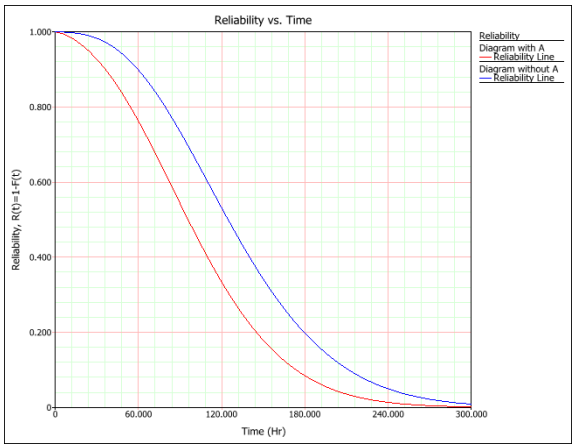


Figure 2.28: A reliability plot example [10]

square root of the number of features (Marslow 2014 [4]). The randomness is key to increase variance without increasing the bias and not needing to prune (Marslow 2014 [4]). The stopping criteria can be, for example an error threshold.

The algorithm works, by aggregating all these trees and applying a majority vote, for classification, or a mean, for regression.

2.5 Reliability

In fault prediction problems, the concepts of reliability, fallibility, hazard and cumulative failure, are very relevant, as such a brief introduction is required.

Reliability, $R_T(t)$, is defined as the probability of performing without failure for a specified period of time, or the probability that at a time t, where $t > 0$, the object of study will function. Another analogous concept is the unreliability, $F_T(t)$, the probability that at a time t, where $t > 0$, the object of study will fail. As such, reliability is defined as:

$$R_T(t) = P(T > t) = 1 - F_T(t) \tag{2.69}$$

where T, is an interval of real numbers.

A Reliability plot is so, something that relates time with the probability of not failing, as can be seen in Figure 2.28. In this figure, it can be seen how with time, in hours, the probability that it will not fail, will diminish, in this case the curve is plotted using the weibull approximation. But the shape of the curve may vary.

2.5.1 Reliability with continuous variables

If one assumes T as continuous random variable, taking values in (0,∞) and with density function $f_T(t)$. The reliability function $R_T(t)$ can be defined as equation (2.70).

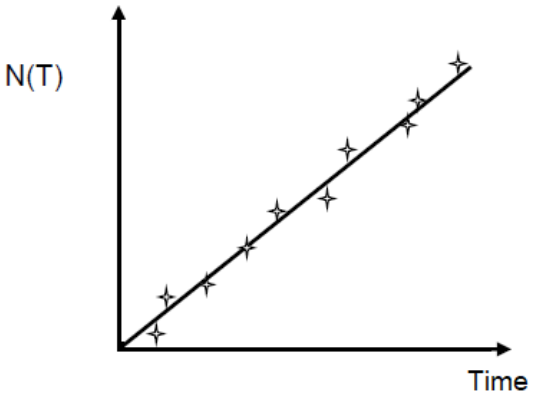


Figure 2.29: Linear trend - failure rate

$$R_T(t) = \int_t^{\infty} f(t)dt \tag{2.70}$$

2.5.2 Failure rate

Failure rate can be defined as the frequency in which a component from an engineered system fails, expressed in failures per unit of time. Failure rate, λ, is a parameter defined as follows in equation (2.71), where $[E\{N(T)\}]$ represents the expected value of the number of failures and T, represents the failure time variable.

$$\lambda(T) = \frac{\partial}{\partial T}[E\{N(T)\}] \tag{2.71}$$

Failures typically can be expected to behave in three different ways.

2.5.2.1 Failure rate - linear trend

Linear trend, this means, Homogeneous Poisson Process (HPP), in here, the failure rate, $\lambda(T)$, is constant and its estimator can be summarized as equation (2.72), where N(T) is cumulative number of failures from time 0 to time T.

$$\lambda(T) \Rightarrow \hat{\lambda} = \frac{N(T)}{T} \tag{2.72}$$

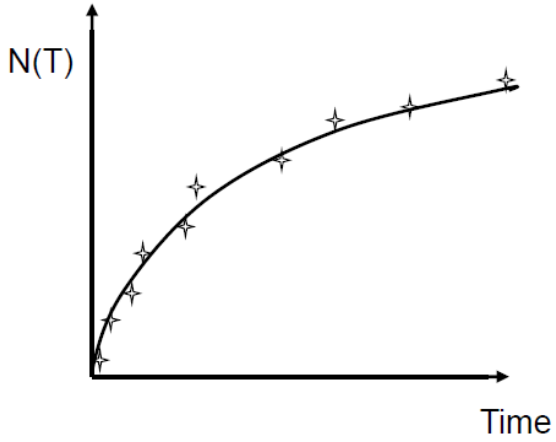


Figure 2.30: Linear trend - failure rate

2.5.2.2 Failure rate - logarithmic trend

Logarithmic trend, this means, the number of failures is decreasing which means reliability growth, in here, the failure rate, $\lambda(T)$, is decreasing and its estimator can be obtained by the model proposed by Crow as equation (2.73).

$$\lambda(T) \Rightarrow \hat{\lambda} = \gamma\beta T^{\beta-1} \tag{2.73}$$

The parameters γ and β, (0<β<1), can be obtained by the maximum likelihood method, proposed by Crow. Where N, is the total number of failures, T_0, total time, t_i, failure time i and T_n, the failure time n.

- Time truncated test (t_0)

$$\hat{\beta} = \frac{N}{\sum_{i=1}^{N} \ln \frac{T_0}{t_i}} \tag{2.74}$$

$$\gamma = \frac{N}{T_0^{\beta}} \tag{2.75}$$

- Failure truncated test (t_n)

$$\hat{\beta} = \frac{N-1}{\sum_{i=1}^{N} \ln \frac{T_n}{t_i}} \tag{2.76}$$

$$\gamma = \frac{N}{T_n^{\beta}} \tag{2.77}$$

2.5.2.3 Failure rate - exponential trend

Exponential trend, this means, the number of failures is increasing which means reliability decay, in this trend, the failure rate, $\lambda(T)$, is increasing and its estimator can be supported by the Weibull distribution, in equation (2.78). The parameters α ($\alpha > 1$) and η can be obtained by the maximum likelihood method.

$$\lambda(T) \Rightarrow \hat{\lambda} = \frac{\alpha}{\eta^{\alpha}} T^{\alpha-1} \tag{2.78}$$

2.5.3 Hazard rate

The hazard function or hazard rate, $h(t)$, for a component is defined as the conditional probability of failure per unit of time at time $(t + \Delta t)$, given it has survived until t. The hazard function in defined by equation (2.79), where $f(t)$, is the probability density function and $R(t)$, is the survival function or reliability.

$$h(T) = \frac{f(t)}{R(t)} \tag{2.79}$$

Another interesting function, is the cumulative hazard function, $H(t)$, defined in equation (2.80).

$$H(T) = \int t - \infty h(t) dt \tag{2.80}$$

As such, reliability can be defined as equation (2.81).

$$R(T) = \exp(-H(t)) \tag{2.81}$$

The hazard rate applied to a component, can be interpreted as, if the hazard rate is decreasing, the component with is becoming less likely to fail, which is not frequent in reality, if the hazard rate is constant, it means the probability of failure is independent of time, and if the hazard rate is increasing, it means it is becoming more likely to fail with time, which is normally, the more common in real life.

The hazard rate can be estimated with parametric models such as Weibull, or non parametric models, such as Kaplan-Meier.

2.5.3.1 Weibull Distribution

The Weibull distribution has a great variety of shapes, therefore it is great in applications for a parametric model, being able to represent all the shapes of the hazard function. It can also be easily handled in the analytical and graphical perspectives. The probability density function, can so be defined as equation (2.82), where β, is a shape parameter, (β>0), η, is characteristic life, which is a scale parameter, (η>0), and γ, is the minimum life, the loction parameter, typically equal to zero, and the distribution function, as equation (2.83).

$$f(t) = \frac{\beta}{\eta^{\beta}} t^{\beta-1} \exp\left[-\left(\frac{t-\gamma}{\eta}\right)^{\beta}\right] with t > 0 \tag{2.82}$$

$$F(t) = \int_0^t f(t)dt = 1 - \exp\left[-\left(\frac{t}{\eta}\right)^{\beta}\right] \tag{2.83}$$

2.5.3.2 Proportional hazards modeling technique

The proportional hazards modeling technique, PHM, seeks to acknowledge several context factors that affect the failure time, t. The PHM, can be defined as a nonparametric technique based on the assumption of a loglinear hazard function which can be applied to assess the effect of observed factors on reliability in engineering applications. The hazard proportional model function can be defined in equation (2.84), where $h_0(t)$, is an arbitrary and unspecified baseline hazard function, z, is the row vector of k measured covariates, factors, β, column vector of k regression parameters, coefficients, and t is associated failure time.

$$h(t;z) = h_0(t)\exp(z_1\beta_1 + z_2\beta_2 + ... + z_k\beta_k) = h_0(t)\exp(z\beta) \tag{2.84}$$

As one can see, the covariates act multiplicatively upon, h_0, which is the baseline hazard function and it is its unique assumption. The main advantages are its distribution free technique, the ability to analyze not independent and identically distributed data, the capacity to deal with sparse non-homogeneous data, the capability to deal with a high level of censoring and the fact that repairable and non repairable systems can also be included in the model.

The survival function can be estimated from the following relationships, present in equations (2.85, 2.86 and 2.87).

$$S_0(t) = \exp\left[-\int_0^t h_0(u)du\right] \tag{2.85}$$

$$S(t;Z) = \exp\left[-\int_0^t h_0(u)\exp(z\beta)du\right] \tag{2.86}$$

$$S(t;Z) = \left[S_0(t)\right]^{\exp(Z\beta)} \tag{2.87}$$

The coefficients, β, can be estimated using the maximum partial likelihood method and the tailor expansion from equation (2.88).

$$L(\beta) = \prod_{i=1}^{n}\left(\frac{\exp(Z(i)\beta)}{\sum_{l\in S(t_i)}^{n}\exp(Z(l)\beta)}\right) \tag{2.88}$$

2.5.3.3 Traditional method

The traditional method is a custom method that had been previously applied to this problem. This method estimations are based in the historical averages of each wind farm, additionally a factor of aging is considered, adding a theoretical failure rate, taken from literature [26]. Therefore, this method is a conservative approach as it adds two failure rates, the wind farm average and the theoretical value. This model can be summarized in equation (2.89), where MC, is the major correctives for the year t, t is the year of the analysis, t_0, the year of commissioning of the wind farm, λ, characteristic life, in years, 27 according to literature [26], k, shape factor, 3,5 according to literature, N, is the number of wind turbines at the wind farm and X is the historic average of major correctives, in other words, the absolute value of the number of components replaced each year.

$$MC = \left[\left(\frac{t-t_0}{\lambda}\right)^{k-1}\frac{k}{\lambda}N\right] + \bar{X} \tag{2.89}$$

2.6 Summary

The field of Machine Learning and Data mining is a field of trade-offs and of not absolute truths. There's not a size fit all algorithm and it depends upon the user to understand the key struggles he faces and apply the model that best addresses them. It is also needed for the data miner to know the tool that best fits the problem, and that requires a deep knowledge of the problem and of the said tools. Reliability is a crucial part of understanding the problem, therefore the concepts are very useful in order to better understand and approach the project. The previous implemented approaches are also substantiated.

Chapter 3

Data collection, preparation and feature selection

As introduced before, the machine learning process, should start with data collection and preparation. First an introduction to the data available, will be given in Section 3.1. Then an analysis of the data will be performed, on Section 3.2, were a search for obvious errors and inconsistencies was performed. The next step, dealing with the missing data and normalization is present, Sections 3.3 and 3.4.4. Then some strategies to deal with idiosyncrasies of this particular data will be introduced, Sections 3.4.2 to 3.4.3.3. All this steps are required as most of the aforementioned algorithms are very sensible to missing data, misrecordings and require normalization. It will also be introduced how inputs were changed in order to better explain the problem. In Section 3.5, the strategy to approach the feature selection, will be expressed.

3.1 Data overview

The data available was collected in wind parks and refers to several wind turbine generators. The data consists of 5314 turbines and 841 fault events. In Table 3.1, all the variables in question are visible, in the overall there are 18.

- "UT" or "Model", identification of each turbine.
- "Park", park identification.
- "Park_Name", name of the park.
- "Platform", the continent were the park is located.
- "COD", data of the park operational start.
- "A", wind scale factor for the park.
- "Air density", the density of the air.

Table 3.1: Data inconsistencies failure

UT	Park_Name	COD	WTG_manufacturer	WTG_model	Failure Date
01 - T001	B1	01/12/2016	V	V1	19/04/2005
01 - T027	B1	01/12/2016	V	V1	03/02/2006
02 - T057	LII	11/05/2008	G	G1	27/03/2008

- "K", wind shape factor for the park.
- "V", average wind speed of the park.
- "LIGHTNING STORM FREQUENCY", the lightning storm frequency for the park.
- "SITE CLASS", the site class for the park.
- "TERRAIN CLASS", the terrain class for the park.
- "DRIVE TRAIN CONFIGURATION", the type of drive train installed in the turbine.
- "WTG_manufacturer", the wind turbine generator manufacturer for a turbine.
- "WTG_model", the wind turbine generator model for a turbine.
- "PRODUCTION NEH", the annual power production estimation for the park.
- "Failure Date", for each event the identification of the turbine and the date of the failure.

3.2 Inconsistency analysis

One of the first inconsistencies present in the data was when checking for the failure times. Here one could clearly see that there were failures that happened before the start of operation of the park. In Table 3.1, are presented some examples, "UT", have a "COD", or date of the park start of operation that happens after the failure, "Failure_date", this clearly was an input error and as such was deleted.

Other input verification that was performed, was to compare the average wind speed, "V", with the power production, as it was clearly apparent, when "V" was zero, the power production could not be bigger than zero, as such those results were flagged as mistakes, see Table 3.2.

Table 3.2: Data inconsistencies power

UT	PRODUCTION NEH	V
01 - T001	2229	0
01 - T002	2229	0
01 - T003	2229	0
01 - T004	2229	0
01 - T005	2229	0

Table 3.3: Data inconsistencies missing data

UT	A	AIR DENSITY	K	LIGHTNING	POWER CLASS
CA - T101	NA	NA	NA	NA	NULL
CA - T102	NA	NA	NA	NA	NULL
CA - T103	NA	NA	NA	NA	NULL
FR - T001	NA	NA	NA	NA	NULL
FR - T002	NA	NA	NA	NA	NULL

3.3 Missing data

Another characteristic of raw data is the absence of data. When confronted with the raw data, it was apparent that there were for several parks, lack of information or data. An example of that in Table 3.3.

3.4 Preparing the data

In order to have results, even if somewhat inevitably skewed results, one had to perform some transformations to the data. If one had decided simply to delete the misrecorded data, or the sections that were null, it would result in a significant diminishment of the failure events, as many of them were relative to turbines with misrecordings or null variables.

Regarding misrecordings, as they were very few, they were replaced with the average of the variable. In the case of Table 3.2, for instance, the wind speed "V", and others were replaced with the average wind speed.

3.4.1 Dealing with missing data

In the case of null variables, the more common cases, as such, they required to be treated with more care. First, there was a difference of treatment if they were categorical, or numeric, for example, if the variable missing is the drive train configuration, or the wind speed.

After careful consideration it was decided that for categorical variables, it would be created a new category which was, the absence of knowledge, or for the drive train configuration example, the null drive train configuration. This implies an obvious data misrepresentation of the data, but was felt that it would not be a very damaging aspect for the overall efficiency of the algorithm.

In order to transform the missing numerical data, the MICE (Multiple Imputed Chain Equations) library of R, was used. Several tests were performed, validating the mean, versus the fast predictive mean matching, versus linear regression, versus imputation by random forests. In the end, it was opted for a mean.

3.4.2 Dealing with categorical data

One of the obvious problems the data had, when face with the algorithms, was that it had several categorical variables, such as "Park_Name", "WTG_model", and others, but most the algorithms

Table 3.4: One-of-n remapping small sample

Park_Name:A	Park_Name:B	Park_Name:C	Park_Name:D	Park_Name:E
TRUE	FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE
FALSE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE

do not work with categorical variables and the ones that do work, prefer numeric or binary variables. As such it was necessary to "dummify" the data, or perform one-of-n remapping. This is the process of turning every categorical variable into a binary variable that states if that variable for those inputs is present, "1" or TRUE, or not, "0" or FALSE, Table 3.4.

3.4.3 Extracting new information

In order to achieve an input and output that was convenient to the problem, several transformations were performed.

3.4.3.1 Dealing with date format

One problem that was identified was the need to transform the failures dates in time intervals, in order to have lifetimes of the turbines. In order to achieve this, the POSIXct format was used. This standard records the seconds since the start of January the first of nineteen seventy. So, now the "COD" and failure dates will be a number. Now simply subtract the "COD", or the date of the park inauguration to the date of the failure and have a time of life, in the case of repeated failures by the same turbine, what is used is the time of the failure minus the time of last failure. In Table 3.5, the data was originally presented is shown, where the time was represented as a date, in the format day/month/year, and now in Table 3.6, there are three columns, the "Failure_Date_original" represents the simple conversion to the POSIXct format, the "Time_Series", is the time of life of the turbine, in other words, it is the time interval, starting from when the turbine was installed to when the turbine failed.

Table 3.5: Time format raw data

Park_Name	COD	Failure Date
A	01/06/2009	18/12/2013
A	01/06/2009	07/03/2015
A	01/06/2009	21/02/2014
A	01/06/2009	20/04/2015

Table 3.6: Time format transformed POSIXct format

Park_Name	Failure_Date_original	Time_Series
A	1392940800	149130000
A	1425686400	181875600
A	1426464000	182653200
A	1429484400	185673600

3.4.3.2 Cumulative failures

Another information that was foreseen as important for the task at hand was the cumulative failures, they represent the amount of failures if time is ordered from the smallest time of life to the biggest. In Table 3.7, there is a small sample of the cumulative failures.

3.4.3.3 Dealing with the Output

The purpose of this paper is to predict the cumulative failures for the next years, using data regarding the previous years. So in this problem it is clear that the desired output is the cumulative failures for the next year.

3.4.4 Using the non failures

The problem that was sought to answer has been addressed in multiple articles, such as [9] [27] [28], but the articles focus on a simple approach, using only the time to failure, cumulative failures and only one technology. In this project, it was sought to use the cumulative failures, the time to failure, multiple technologies, but also different factors that, were relevant to the the life span of turbine, such as average wind speed, the type of terrain, the amount of power produced, and others. In order to do so, it was fed to the network the data referring to all the turbines, combined with failures and non failures, if for example a turbine hadn't failed yet, it was used that information, feeding to the network the time of last record as the time to fail, but the cumulative failure would remain constant (Table 3.8). The last recorded observation was in the 31st of December of 2014, due to the fact that it was defined that 2015 would be the testing year, as such the information of non failure is also present.

Table 3.7: Cumulative failures example

Park_Name	Platform	COD	Time_Series	Cuml_Fail
A	A	1243810800	143514000	1
A	A	1243810800	149130000	2
A	A	1243810800	181875600	3
A	A	1243810800	182653200	4
B	A	1114902000	242614800	1
C	A	1370041200	26438400	1

3.4.5 Normalizing the data

The formula used to normalize the data was the aforementioned, equation (3.1).

$$x_{n_N} = \frac{x_n - min(x_1 ... X_N)}{max(x_1 ... x_N) - min(x_1 ... x_N)} \tag{3.1}$$

Regarding the cumulative failures and the time series, the normalization posed some constraints, because the maximum value is unknown. It was arbitrated that forty years was a time interval large enough to include all the life expectancy of a turbine, as such, in order to normalize the cumulative failures, the maximum number of turbines of a park was used as the maximum, and zero as the minimum, for the time series, the largest time interval was used as the maximum and zero was used as the minimum.

3.5 Feature selection

hat is sought in feature selection is to select relevant and informative features, in order to achieve several different objectives, namely general data reduction, to decrease the amount of data storage required and increase the speed of the algorithm, Feature set reduction, save resources in the next round of data collection or during utilization, allows for performance improvement, gains predictive accuracy and establish a better data understanding.

3.5.1 Understanding the data

In order to select the best features and before implementing sophisticated methods, one should analyze the data. It is clear that some of the inputs are dependent on one another, or are simply different ways of stating the same information and as such redundant.

Some inputs have exactly the same information, for example "Park" and "Park_Name", "Park" is merely a tag for the park, as such it is redundant. Other auxiliary variables that were created upon processing the data can also be discarded.

The input "Model", and the "Park_Name" are cases where it would have a disrupting impact in the model,as they are all encompassing and would undermine the ability of the algorithm to generalize.

3.5.2 Filtering the data

It was an objective to try and solve the problem that this thesis addresses by compartmentalized steps, in other words, the goal was to not have to, at this step, choose an algorithm and have a compartimentalized approach to the problem. It was decided that a filter was more appropriate, as a *filter* enables the separation between the feature selection and the algorithm, whereas an *Embedded* or a *wrapper*, Section 2.3.6.1, method would be highly dependent on the chosen algorithm.

3.5.3 Choosing the data

It was decided to try the ReliefF, Section 2.3.6.2, filters, in order to ascertain if all the features are worth considering or if some can be discarded. The ReliefF showed some differences in importance in the data, but no feature was deemed irrelevant, so it was decided to use all the features that remained. The computational speed for this thesis deemed as secondary objective.

3.6 Summary

A determinant element in the performance of an algorithm is molding the data, and molding it to the answers the the problem that presents itself. In this approach it is key, to understand the data, address it is limitation and adapt it to the algorithm. The key steps of preparing the data were, disposing of errors, addressing data absences, manipulating categorical data and normalizing. Feature Selection is important to trim the data into and shed redundant information. It was used a mixture of data interpretation mixed with the use of an analysis using the relief algorithm.

Chapter 4

Algorithm choice, parameter and model selection

As introduced before, the world of machine learning is rich with different algorithms, that have different objectives and different strengths and weaknesses. As such, now that a dataset is had, what algorithm to use and what specific traits have to be chosen, as most of them have parameters that require manual adjustment. First in Section 4.1,the choosing of the algorithm will be discussed, Section 4.2 will explore how to best tune the algorithms parameters to generate the best network.

4.1 Primary objectives

It is important to understand what is relevant to this problem and what type of problem it is. In this case, the problem is a regression and a case of supervised learning. It is a regression, due to the fact that the goal is not to classify the problem into pre-defined classes, but to "discover a predictive learning function, which maps a data item to a real-value prediction variable" [1](Figures 2.8 and 2.9). It is a supervised learning problem, since there is a set of known failures and network trains using that set for an error comparison (Figure 2.3).

So now the algorithm that best suits the needs of the problem, has to be one that best works with regressions and supervised problems. Other criteria can be contemplated, for example, some algorithms have an emphasis on comprehensibility to the user and others on computational speed. For this problem, the time interval is of years, as such the computational speed has little relevance, as this project is not to be interpreted by other users that are not familiar with this topics, the comprehensibility aspect is also of little impact. Therefore, the main focus of the algorithm will be accuracy.

4.2 Choosing an algorithm

The algorithm choice is not a straightforward choice, most of them have strengths and weaknesses and behave differently with the size of the dataset and the objective. The decision trees have a number of disadvantages that make them a lesser candidate from the start, even though they are easy to interpret, they are not very robust (Hastie et al. 2001 [25]), it splits in accordance to the axis which can be very suboptimal (Bishop 2006 [6]), so typically a decision tree sub performs when compared to other algorithms. When deciding between a support vector machine and a neural network, the arguments are more even, but due to the support vector machines being more common to classification problems and to neural networks being a more common and widespread technology, it was chosen to go with neural networks.

4.3 Parameter and model selection

When dealing with neural networks, some parameters must be manually defined. Using a traditional feed forward network, one must define:

- Number of hidden layers
- Number of hidden neurons
- Activation Function
- Error Function
- Training Algorithm

In order to have a non-linear network, one needs at least one hidden layer, but an addition of a few more, can better read the non linearities of the problem.

In order to choose the amount of hidden neurons a definitive version has not been decided, yet some general rules prevail, such as that the number of neurons must be between the number of outputs and inputs. It was arbitrated to start using a generic formula, equation (4.1), where N_s is the number of observations, N_i is the number of input neurons, and N_o is the number of output neurons and alpha is equivalent to a degree of freedom, arbitrated to be between 2 and 10. This values are just references and are optimized by testing the algorithm, but they give a useful interval.

$$N_h = N_s/(\alpha * (N_i + N_o)) \tag{4.1}$$

The activation function typically is either a logistic function or a tan-h function, as explained in Section 2.4.4.2, as the output varies between 0 and 1, and not between -1 and 1, a logistic function is more appropriate. The error function to access the accuracy will be the sum of squared errors.

The training algorithm opens several options namely the discussed above resilient backpropagation and the traditional backpropagation, which requires a learning rate. It was arbitrated to use resilient backpropagation, due to being faster and at least as good as backpropagation, as explained in Section 2.4.4.7.

Chapter 5

Training, evaluation and interpretation of the results

This chapter will show the results and the results of training the network, and make a comparison between different models. It will try to present the results in the two errors, generalization error, Section 5.1 and Section 5.4, and prediction error, Section 5.1.2 and Section 5.5. This chapter will also offer a comparison between the different models and their performances, Section 5.5.

5.1 Optimization criteria

To evaluate the algorithm, the dataset was split into two parts, one referring to the year 2015 and the other referring to the period before 2015, as this year will be used to control and optimize the prediction error. Then the data will be split in a 10/90 ratio, where 90 percent of the data will be used to train and 10 percent will be used to test the network, creating a validation set, for overfitting, ensuring the network is generalizing well, therefore this error is called generalization error. This can be seen in Figure 5.1.

5.1.1 Preventing overfitting

To prevent overfitting the network, an error threshold criteria and maximum steps criteria, plus the generalization error, bellow described, were defined.

The equation (5.1), refers to the mean squared error for the generalization error, e^g, where $Pred$, is the prediction of the network given the inputs referring to the validation set, that the network has never seen, $Actual$ is the true value, given the inputs of the validation set, and N_v is as the total number of samples for the validation set. In equation (5.2), the root mean squared error for the generalization error, Re^g.

$$e_t^g = \sum_{i=1}^{N_v} (Pred_i - Actual_i)^2 / N_v \tag{5.1}$$

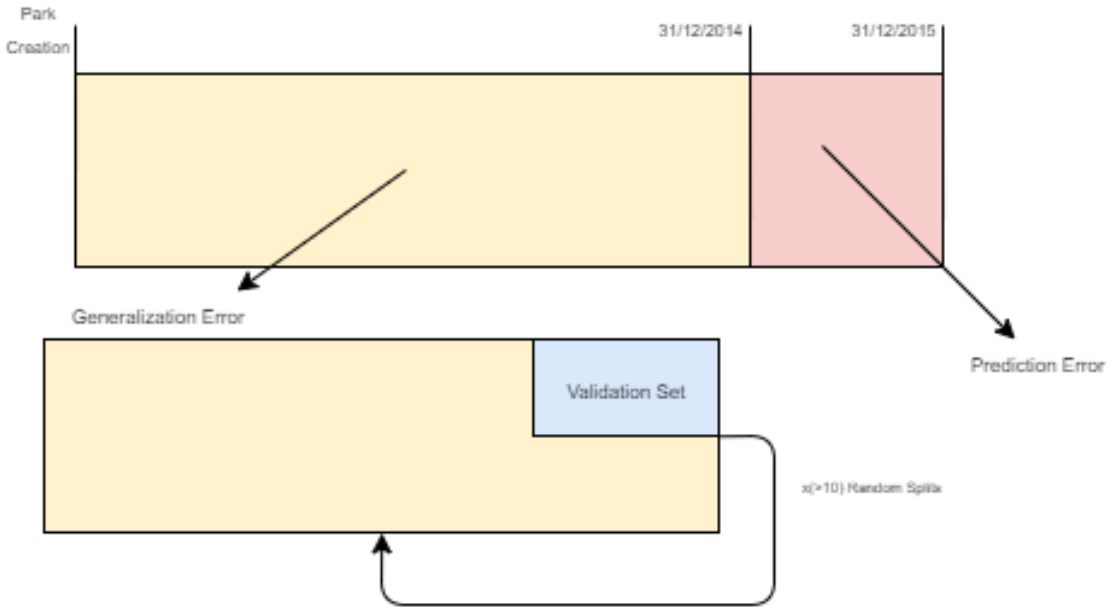


Figure 5.1: Description of the data splits

$$Re^g = \sqrt[2]{\sum_{i=1}^{N_v} (Pred_i - Actual_i)^2 / N_v} \tag{5.2}$$

Furthermore, in order to better generalize and to ensure the best network, the data will be split randomly in the above mentioned 10/90 ratio at least 10 times, in order to find the split that creates the best network. After the training iterations, the network that best optimizes the two errors is saved, and the others discarded.

5.1.2 Prediction Error

The equation (5.3), is the mean squared error, MSE, for the prediction error, e^p, if one considers $Actual\_2015$, as the value regarding to the true number of faults for a given set of inputs in the year of 2015, $Pred\_2015$ as the value pertaining to the algorithm prediction for given a set of inputs for the year of 2015 and N_s as the total number of samples for the year 2015. In equation (5.4), the root mean squared error for the prediction error, Re^p.

$$e^p = \sum_{i=1}^{N_s} (Pred\_2015_i - Actual\_2015_i)^2 / N_s \tag{5.3}$$

$$Re^{p} = \sqrt[2]{\sum_{i=1}^{N_s} (Pred\_2015_i - Actual\_2015_i)^2 / N_s} \tag{5.4}$$

5.2 Network configuration

As explained in Section 4.3, some parameters must be manually adjusted. The criteria to select the best parameters was optimizing the errors, by adding both.

Therefore, the variations per iteration will be the α in the hidden neuron numbers, using 2, 5, 10, and the number of hidden layers, between 2,3 and 4.

Table 5.1: Algorithm Iterations

N^{o} Hidden Layers	N^{o} Hidden Neurons	Generalization Error (RMSE)	Prediction Error (RMSE)
2	2	0.008040232	0.87124
	5	0.010328135	0.91782
	10	0.027753364	1.00235
3	2	0.007375599	0.93195
	5	0.008086259	0.88025
	10	0.018754856	0.99399
4	2	0.000751648	0.72066
	5	0.011997014	0.92231
	10	0.006599443	1.14014

As seen in Table 5.1, the one that minimizes the error of prediction is the one with an α of 2 and 4 hidden layers.

5.3 Network architecture

As demonstrated in Section 5.2, the best neural network architecture used was a neural network with one output node, 84 input nodes, four hidden layers, and 7 hidden neurons, applying an alpha of 2, and using the equation (4.1).

5.4 Generalization error results

This error as explained in Section 5.1.1 and Section 2.4.3, equation (5.1), pertains to the ability of the network to generalize and to not overfit. The ability to generalize is the ability for a network to learn the network and establishing weights, without memorizing results, which would at first glance seem to improve performance, but in fact be making the network unable to give accurate results to new inputs and never seen inputs. The root mean squared error, RMSE, of the generalization error was used as the optimization criteria.

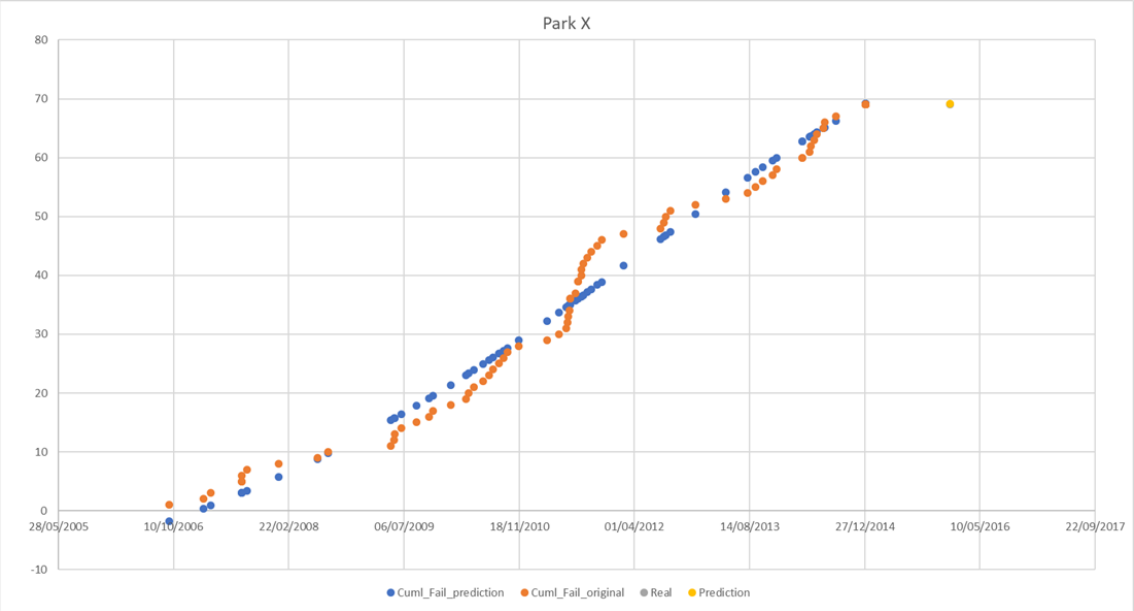


Figure 5.2: Network performance by park - large number of failures

In Figures 5.2 and 5.3, the network is not overfitting. Both figures show the network prediction for a park X, with a medium number of failures and a park Y, with a large number of failures. On the horizontal axis is the time, in the vertical axis the cumulative failures. In orange, "Cuml_Fail_original", the real values, in blue the corresponding predicted values, in gray, the real cumulative failures value for the year 2015, the year used to check accuracy, and in yellow the corresponding prediction, for the cumulative failures for the year 2015.

For the park Y, is clear that the prediction worsens. If there are less data points and they are less homogeneously spread the model accuracy diminishes, on the other hand for the park X, as there are more data samples, the prediction almost overlaps with the real value and the prediction values for the data previous to 2015 are nearer. The graphic seems to be linear and then turns logarithmic, this happens due to the factoring of the non failures, as explained in Section 3.4.4, the non failures are introduced with the time corresponding to the last known record and they have the cumulative failure of the last cumulative fail registered, as such they attenuate the effect of having few fail data, for a given park.

5.5 Comparing results

The goal of this thesis was always to compare the proposed methods against other already implemented algorithms, as such in this section one will compare the results above mentioned with the results that had already been processed for the other methods.

The prediction error is the error related to this thesis ultimate goal, creating a network that best predicts failures. This error is the predictions the network makes for the values referring to the year 2015, that were separated before training, as such this is new data that the network has never seen.

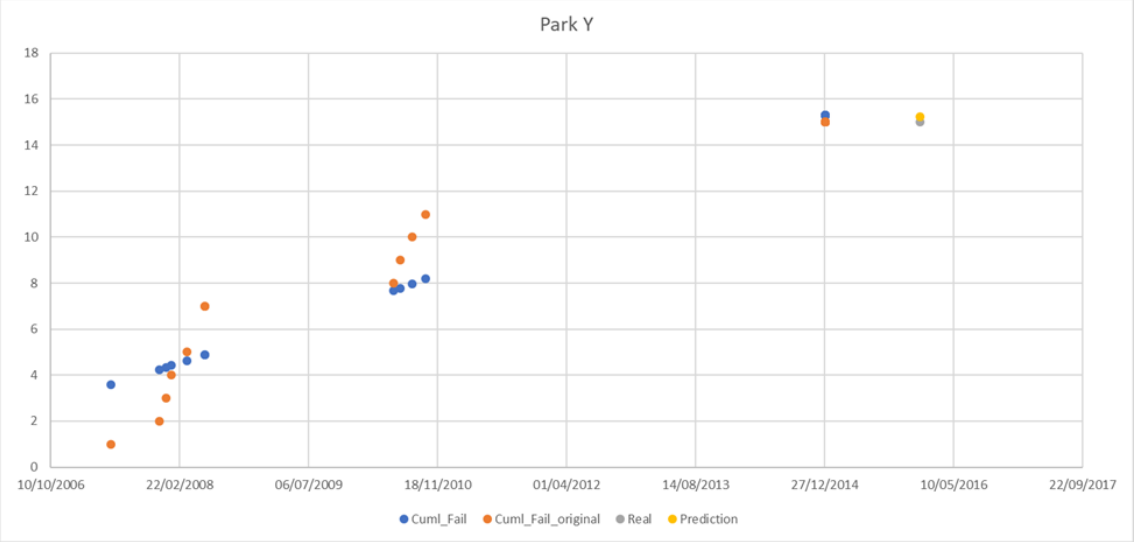


Figure 5.3: Network performance by park - medium number of failures

So what was done was give those inputs to the network and compare them with the corresponding real values. The traditional method is as defined in Section 2.5.3.3 and the proportional hazards model, PHM, is as defined in Section 2.5.3.2.

Table 5.2: Algorithm predictions error by park

Traditional Prediction RMSE	PHM Prediction RMSE	Neural Network Prediction RMSE
0.66839	0.795869	0.72066

In Table 5.2, the RMSE for each algorithm varies, the lower the error the better the model can fit reality, observing the table it can be concluded that the traditional model, is the one that best fits the network, being followed by the neural network and the PHM. Nevertheless, other factors must be taken into account and that is what it will be done in the rest of the section.

Table 5.3: Error prediction by park group in percentage, T, stands for the traditional method, PHM, stands for the proportional hazards model and NN, for the neural network.

Interval	Park%	T Avg error%	PHM Avg error%	NN Avg error%
Very Small	39%	10.12%	9.75%	11.69%
Small	24%	4.13%	4.54%	4.93%
Medium	16%	3.21%	3.37%	2.14%
Large	21%	1.55%	2.24%	1.68%

In Table 5.3, the error analysis can be more detailed, the parks were divided into four categories, very small, small, medium and large, referring to the number of turbines. As expected, the error is greater for smaller parks, as they have less information and less occurrences, and as

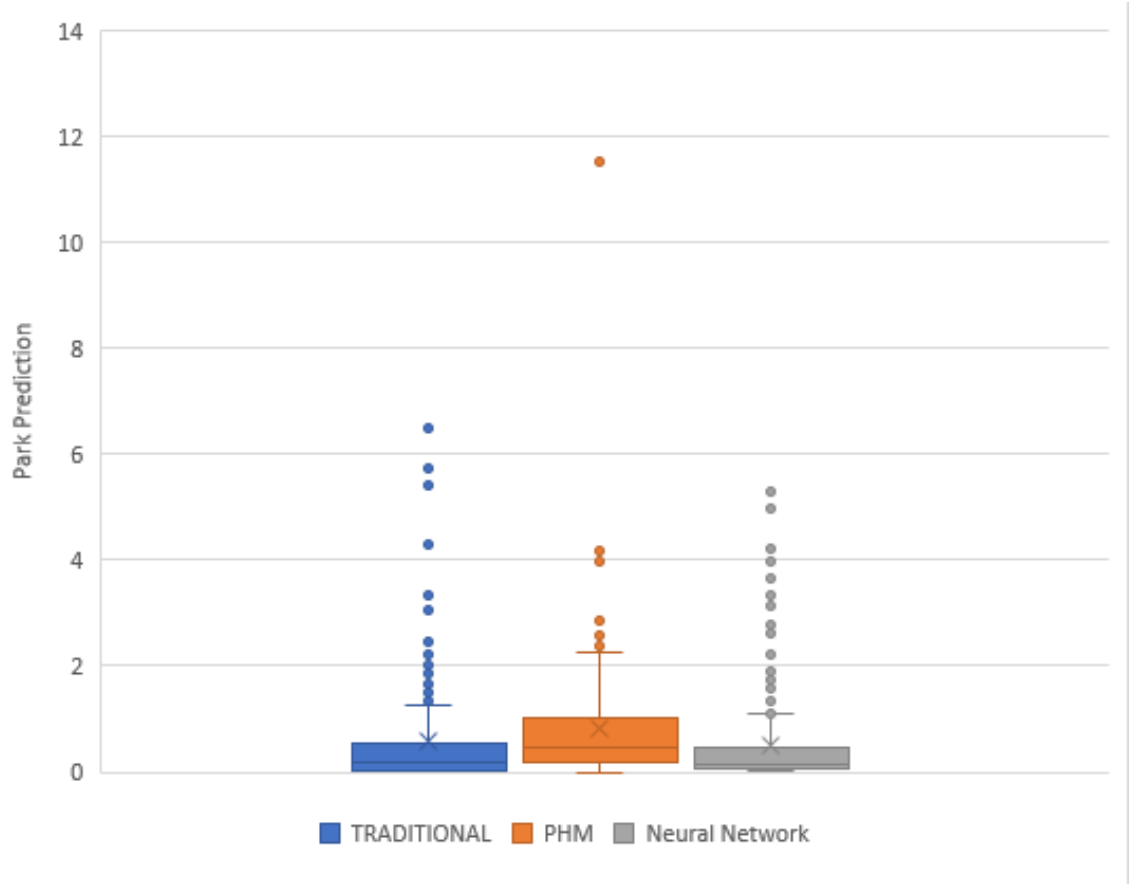


Figure 5.4: Error dispersion

the size of the park increases, the error prediction becomes more accurate. The methods behave differently according to the size of the park, the traditional method achieves better results than the other for, small and large parks, the PHM method, has better results for very small parks, and the neural network has better results for medium parks.

Figure 5.4, presents the box plot for the prediction of failures by park, for the year 2015. In this figure, observe that the neural network, in gray, has the least range on its predictions, and that its outliers are also the least spread, regarding the traditional method, it can be verified that it has the second smallest range and the second smallest spread of its outliers, finally, the PHM model has the biggest range and the fewer outliers, even though its outliers have the biggest spread.

5.5.1 Comparing results by technology

As discussed in the motivation, Section 1.2, the technology is a key part of optimizing costs. As interesting as knowing failures per park, the number of failures per technology is also a key indicator, due to the costs being related to technology, for instance the amount of components needed per year is a crucial information, to manage spare parts stock levels.

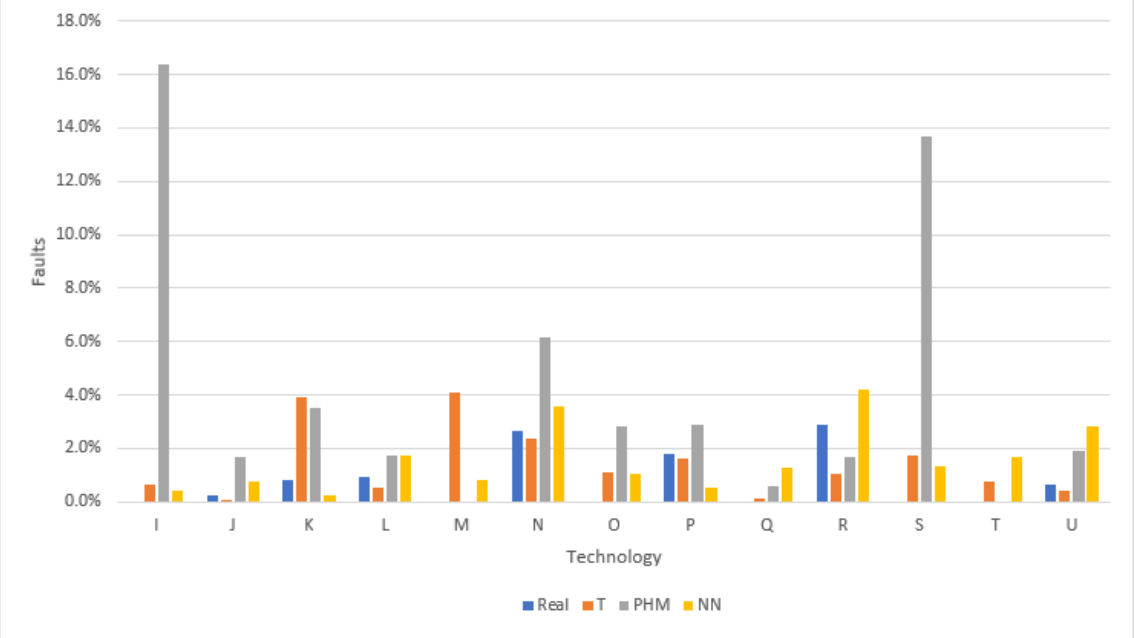


Figure 5.5: Error prediction comparison by technology

If the results are grouped by technology, a similar trend, in Table 5.4, the order of the best results is the same being the tradition the best method, the neural network the second best and the model based on the PHM is the worst.

Table 5.4: Algorithm predictions error by technology

Traditional Prediction RMSE	PHM Prediction RMSE	Neural Network Prediction RMSE
2.44	3.0194	2.549

In Table 5.5, the results of each different technology are differentiated, it was used a simple rank system, known as the M. Friedman's statistic average ranking. The rank corresponds to the best error, and varies between 1, the best error, and 3, the worst error. The results match up with the RMSE, being that the best is the traditional method, the worst the PHM and in second the neural network. In bold, the best result for each park. In the table also see how the neural network and the traditional method, for bigger errors tend to undershoot, if one takes in consideration technologies, "Y,AA,AB,AC,AD,AE,AH and AJ", this effect seems clear, regarding the PHM, no such trend can be inferred.

Complementary, in Figure 5.4, the data grouped by technology on the horizontal axis, and the vertical axis, the predicted number of failures by technology. In orange, the traditional method, in gray, the PHM method, in yellow the neural network method and in blue the real number of failures by technology.

5.6 Summary

The root mean squared error enables us to state that the traditional method is still better, by park and technology and by only technology. The neural network, performs better than the PHM in park and technology and by only technology.

Table 5.5: Error prediction by technology in percentage, T, stands for the traditional method, PHM, stands for the proportional hazards model, NN, for the neural network, Tech for technology and the R stands for rank.

Tech	Actual	T%	PHM%	NN%	Δ T%	Δ PHM %	Δ NN%	R T	R PHM	R NN
A	0.0%	0.0%	5.6%	0.0%	0.03%	5.6%	**0.01%**	2	3	1
B	0.5%	0.0%	1.5%	0.6%	0.5%	0.9%	**0.0%**	2	3	1
C	1.5%	7.3%	2.5%	1.3%	5.8%	1.0%	**0.2%**	3	2	1
D	0.0%	0.0%	0.7%	0.2%	**0.0%**	0.7%	0.2%	1	3	2
E	0.0%	17.5%	20.2%	0.2%	17.5%	20.2%	**0.2%**	2	3	1
F	0.0%	0.0%	0.5%	0.3%	**0.0%**	0.5%	0.3%	1	3	2
G	0.0%	0.1%	0.5%	0.3%	**0.1%**	0.5%	0.3%	1	3	2
H	0.5%	3.2%	6.0%	0.2%	2.7%	5.4%	**0.4%**	2	3	1
I	0.0%	0.6%	16.4%	0.4%	0.6%	16.4%	**0.4%**	2	3	1
J	0.2%	0.0%	1.7%	0.8%	**0.2%**	1.5%	0.5%	1	3	2
K	0.8%	3.9%	3.5%	0.3%	3.1%	2.6%	**0.6%**	3	2	1
L	1.0%	0.5%	1.7%	1.7%	**0.4%**	0.8%	0.8%	1	3	2
M	0.0%	4.1%	0.0%	0.8%	4.1%	**0.0%**	0.8%	3	1	2
N	2.6%	2.4%	6.1%	3.6%	**0.3%**	3.5%	1.0%	1	3	2
O	0.0%	1.1%	2.8%	1.0%	1.1%	2.8%	**1.0%**	2	3	1
P	1.8%	1.6%	2.9%	0.6%	**0.2%**	1.1%	1.3%	1	2	3
Q	0.0%	0.1%	0.6%	1.3%	**0.1%**	0.6%	1.3%	1	2	3
R	2.9%	1.0%	1.7%	4.2%	1.8%	**1.2%**	1.3%	3	1	2
S	0.0%	1.7%	13.7%	1.3%	1.7%	13.7%	**1.3%**	2	3	1
T	0.0%	0.7%	0.0%	1.7%	0.7%	**0.0%**	1.7%	2	1	3
U	0.6%	0.4%	1.9%	2.8%	**0.2%**	1.3%	2.2%	1	2	3
V	3.2%	0.0%	0.4%	0.6%	3.2%	2.9%	**2.6%**	3	2	1
X	3.2%	2.6%	2.5%	0.4%	**0.6%**	0.8%	2.8%	1	2	3
Y	4.4%	2.8%	6.7%	7.2%	**1.6%**	2.3%	2.8%	1	2	3
Z	0.0%	1.0%	3.5%	2.8%	**1.0%**	3.5%	2.8%	1	3	2
AA	5.0%	21.6%	16.6%	1.0%	16.6%	11.6%	**4.0%**	3	2	1
AB	5.2%	7.1%	5.5%	1.1%	1.9%	**0.3%**	4.1%	2	1	3
AC	3.1%	3.6%	5.9%	7.9%	**0.5%**	2.8%	4.8%	1	2	3
AD	7.2%	3.3%	4.9%	1.8%	3.9%	**2.4%**	5.4%	2	1	3
AE	37.0%	9.1%	15.3%	31.6%	27.9%	21.7%	**5.5%**	3	2	1
AF	0.0%	0.4%	0.0%	5.8%	0.4%	**0.0%**	5.8%	2	1	3
AG	0.0%	8.6%	10.7%	6.2%	8.6%	10.7%	**6.2%**	2	3	1
AI	0.0%	10.5%	12.4%	8.3%	10.5%	12.4%	**8.3%**	2	3	1
AH	9.1%	3.3%	10.4%	17.7 %	5.8%	**1.3%**	8.6%	2	1	3
AJ	15.9%	10.7%	7.7%	0.9%	**5.2%**	8.2%	15.1%	1	2	3
							Avg Rank	1.80	2.26	1.94

Chapter 6

Conclusion and future work

This chapter will offer conclusions and possibilities for future improvements. Moreover, it also suggests how the work performed is relevant and its performance.

6.1 Objectives satisfaction

The main goal was to use a machine learning algorithm that would predict failures of the turbines. This would help to minimize operation and maintenance costs, as it would offer a better insight and time frame, for allocating maintenance resources. The challenge was to compare a machine learning algorithm against two methods that had been already employed and to which data for predictions had already been provided. In this scope, a neural network was chosen to operate as the algorithm of machine learning. What could be seen was that the neural network was able to perform better than the PHM model which was the main objective, as such it could be deemed as a success.

That said, the network still suffered a lot of variation. This can be attributed to the fact that for some turbines, there was not much useful information, creating data gaps, and the fact that the turbines had not completed a life cycle and in fact, some were in the beginning of their life cycle, making predictions harder, and the behavior less stable. Nevertheless, the algorithm was able to predict failures, with a what seemed a reasonable error.

The incrementation upon the work performed was the ability to implement an algorithm that offered less constraints and assumptions compared to the PHM model. That was what drove the work, the possibility to create a network that would offer better and more accurate results, even when confronted with non linear life cycles.

6.2 Future work

The increase in information on a day by day basis, produces more data and as such enables the network to better fit and better predict future failures. As such, keeping expanding the network with more results inputs is in itself an interesting element to the future. It would be beneficial

if the network would at least acknowledge a complete life cycle for all the technologies, but as discussed in Section 1.1, the turbines life cycle are rather long, taking up to twenty years.

An interesting increment would be to implement this level of predictive maintenance to all the key components of the turbine. This way one could have the whole map of behavior of the turbine cycle of life and as such produce more reliable predictions. This would also help to understand how different components interact and behave through time.

Another curious increment, would be to intertwine the prediction with the expected cost, in other words, to adjust the pessimism or optimism of the prediction model to the expected cost of a failure not predicted or for a failure that was expected and did not happen. This would force a more in depth understanding of maintenance costs. This would help us not obtain simply a network that minimizes the error of the prediction, but a network that minimizes the cost of maintenance, turning this work into a more useful work to the community. This is ultimately the end goal to offer an algorithm that had a tangible impact in reducing costs and improving performance.

Furthermore, implementing this logic to other algorithms would be interesting in order to compare algorithm performance.

References

[1] Mehmed Kantardzic. *Data Mining: Concepts, Models, Methods,and Algorithms*. Wiley-IEEE Press, second edition, 2011.

[2] Emma Brunskill. Cs234: Reinforcement learning, June 2017. URL: http://web.stanford.edu/class/cs234/index.html [last accessed 2017-06-20].

[3] Walter Pitts Warren S. McCulloch. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics , Volume 5, Issue 4,*, page 115–133, December 1943.

[4] Stephen Marsland. *MACHINE LEARNING An Algorithmic Perspective*. Chapman and Hall CRC, second edition, 2014.

[5] Sebastian Raschka. Single-layer neural networks and gradient descent, June 2017. URL: https://sebastianraschka.com/images/blog/2015/singlelayer_neural_networks_files/perceptron_learning_rate.png [last accessed 2017-06-20].

[6] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

[7] Michael I. Jordan. Serial order a parallel distributed approach. May 1986.

[8] Yashwant Malaiya Nachimuthu Karunanithi, Darrell Whitley. Using neural networks in reliability prediction. *IEEE Software 9(4)*, pages 53–59, August 1992.

[9] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery archive Volume 2 Issue 2*, pages 121 – 167, June 1998.

[10] ReliaSoft Corporation. Reliability importance measures of components in a complex system - identifying the 20 URL: http://www.weibull.com/hotwire/issue66/rb66-16.png [last accessed 2017-06-20].

[11] Amparo Alonso-Betanzos Verónica Bolón-Canedo, Noelia Sánchez-Maroño. *Feature Selection for High-Dimensional Data*. Springer Berlin Heidelberg, first edition, 2012.

[12] André Elisseeff Isabelle Guyon. *An Introduction to Feature Extraction*. Springer Berlin Heidelberg, first edition, 2006.

[13] Igor Kononenko Marko Robnik-Šikonja. Theoretical and empirical analysis of relieff and rrelieff. *Machine Learning October 2003, Volume 53, Issue 1,*, pages 249–256, October 2003.

[14] Tom M. Mitchell. *The Discipline of Machine Learning*. School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213, first edition, 2006.

[15] Dorian Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann Publishers, first edition, 1999.

[16] Sergios Theodoridis Konstantinos Koutroumbas. *Pattern Recognition*. Elsevier inc, first edition, 2008.

[17] Larry A. Rendell Kenji Kira. A practical approach to feature selection. *Proceeding ML92 Proceedings of the ninth international workshop on Machine learning*, pages 249–256, 1992.

[18] Igor Kononenko. Estimating attributes: Analysis and extensions of relief. *Machine Learning: ECML-94. ECML 1994. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence), vol 784.*, 1994.

[19] David G. Stork Richard O. Duda, Peter E. Hart. *Pattern Classification*. Springer Berlin Heidelberg, second edition, 2001.

[20] Padhraic Smyth David Hand, Heikki Mannila. *Principles of Data Mining*. MIT Press, 2001.

[21] Jeff Schneider. Cross validation, June 2017. URL: https://www.cs.cmu.edu/~schneide/tut5/node42.html [last accessed 2017-06-20].

[22] RONALD J. WILLIAMS DAVID E. RUMELHART, GEOFFREY E. HINTON. Learning representations by back-propagating errors. *Nature 323*, pages 533 – 536, October 1986.

[23] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS*, pages 586–591, 1993.

[24] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM. URL: http://doi.acm.org/10.1145/130385.130401, doi:10.1145/130385.130401.

[25] Jerome Friedman Trevor Hastie, Robert Tibshirani. *The Elements of Statistical Learning Data Mining, Inference, and Prediction,*. Springer-Verlag New York, second edition, 2009.

[26] R. Poore and C. Walford. Development of an operations and maintenance cost model to identify cost of energy savings for low wind speed turbines. *Report for the National Renewable Energy Laboratory of United States*, June 2008.

[27] L.C. Tang S.L. Ho K. Xu M. Xie. Application of neural networks in forecasting engine systems reliability. *Applied Soft Computing Volume 2, Issue 4,*, pages 255–268, February 2003.

[28] Huan-Jyh Shyur James T. Luxhoj. Reliability curve fitting for aging helicopter components. *Reliability Engineering and System Safety Volume 48, Issue 3,*, pages 229–234, 1995.